Escuela Politécnica Superior

Desarrollo de un J-RPG en Unity

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Ricardo Valls Luna

Tutor/es:
Miguel Ángel Lozano

Septiembre 2017

Universitat d'Alacant
Universidad de Alicante

CITAS

"*You get talent when you work on something and get good at it*"
Edmund McMillen
Desarrollador de Super Meat Boy y The Binding of Isaac

DEDICATORIA

Han sido 5 años en total en realizar la titulación, donde he pasado por una gran cantidad de experiencias que me han hecho crecer, sobre todo a nivel personal.

Para comenzar quiero agradecer a mi tutor, Miguel Ángel, toda la paciencia que habrá tenido que tener conmigo y la ayuda que me ha brindado a lo largo del desarrollo de este trabajo.

También quiero agradecer a las grandes amistades que he hecho en esta titulación, sin ellos no habría sido lo mismo. Gracias a ellos he podido seguir adelante en muchos momentos. En especial me gustaría nombrar a mis compañeros de ABP: Carlos, Alex, Alberto y Albert que junto con Alejandro han estado ahí tanto para las bromas y las risas como en los momentos más duros.

Finalmente me gustaría darle las gracias a mi padre, es gracias a él que he podido estudiar en la universidad. Sin su apoyo, su fe en mí pese a mis fallos y su esfuerzo nada de esto habría sido posible.

RESUMEN

El mundo de los videojuegos ha evolucionado y crecido mucho en los últimos años dando a conocer nuevos géneros, nuevos gráficos, etc. Pero hay un género en concreto que es considerado prácticamente, de culto. Estamos hablando del RPG (Rol Playable Game), un género que existe desde los comienzos del videojuego y que ha extendido su legado hasta el día de hoy.

Pese a que el género como tal ha evolucionado bastante, en este proyecto se ha desarrollado un videojuego jrpg en Unity con presupuesto nulo. Para el proyecto se ha perseguido ese toque clásico que tenían antaño los jrpg, con gráficos 2D, combates por turnos y sistema de trabajos para los personajes.

INDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

Los videojuegos están en su época dorada, hoy en día están más aceptados que nunca dentro de la sociedad donde los jugadores de videojuegos ya no son considerados unos "raritos". Los videojuegos hoy en día son fenómenos que afectan a miles de personas, sobre todo los videojuegos móviles, los cuales tienen más efecto en la sociedad (Pokemon Go es un ejemplo de esto). Y en el caso de los juegos de grandes compañías, los famosos juegos "AAA" son producciones que están a la altura de producciones cinematográficas debido a su alto presupuesto.

Con el paso del tiempo los videojuegos han ido evolucionando y han surgido nuevos géneros, nuevas ideas, nuevas mecánicas… Pero dentro del público siempre queda el recuerdo de su primer videojuego, y es donde **Ivankar's Tale** tiene lugar, pues su escenificación "retro" con gráficos en 2D y su jugabilidad de aquellos primeros juegos de rol incitan a una vuelta al pasado para recordar aquellos tiempos.

El documento está estructurado de una forma concreta para facilitar su lectura y comprensión.

Primeramente el *Marco teórico*, donde se hace un pequeño estudio del género rpg (junto con sus subgéneros), los motores de videojuegos que hay en el mercado y los juegos que han servido de inspiración y referencia para realizar este proyecto.

Seguidamente se encuentra el apartado de *Objetivos* donde se describen detalladamente los objetivos que se pretenden conseguir con este trabajo.

A continuación, el apartado de *Metodología*, donde se describe la metodología usada así como el control de versiones utilizado en el trabajo.

En el apartado de *Cuerpo del trabajo* se encuentra el documento de diseño del juego (GDD), una lista de los requisitos funcionales y no funcionales, recursos utilizados (herramientas, arte del juego) y todo aquello relacionado con la implementación.

Después en la sección de *Conclusiones* aparecen mis opiniones acerca del desarrollo del proyecto, así como todo lo aprendido en su realización.

Por último en el *Anexo: Diagrama de clases* se puede encontrar una explicación más detallada de las clases utilizadas para el desarrollo de este trabajo.

2. Marco Teórico o Estado del arte

En este apartado se va a hacer un pequeño estudio acerca de los videojuegos, en concreto de los videojuegos jrpg: ¿qué es? ¿qué variantes hay?... Además de este estudio, también se va a hacer otro estudio en relación a los motores de videojuegos en el mercado actual para ver que opciones y alternativas ofrecen.

2.1 Definición de RPG

RPG es el acrónimo de Rol Playable Game, en otras palabras, juego de rol.

Un rol es un papel, representación o interpretación de un personaje de forma imaginaria o interpretativa.

2.2 Videojuegos RPG

El RPG como videojuego es aquel en el que el jugador encarna a uno o varios personajes que atraviesan una serie de situaciones en las que el jugador tiene la capacidad de decidir (o no) el transcurso de la historia.

Las características principales de estos videojuegos vienen dadas por la adaptación de algunos elementos de los juegos de rol tradicionales o de mesa como por ejemplo, el desarrollo estadístico del personaje.

Normalmente, dentro del género RPG, los personajes controlados por el jugador pueden acumular puntos de experiencia y de esta forma, al llegar a un número determinado de puntos (50, 100,…), subir de nivel (junto con sus estadísticas). A su vez, los jugadores pueden conseguir dinero e intercambiarlo por bienes para su causa como pociones, comidas, armas, armaduras, habilidades,…

Actualmente, predomina la propuesta de videojuego donde se controla y representa cabalmente a un personaje (o varios), que debe cumplir con una serie de objetivos o misiones establecidos. Usualmente, se crea un mundo perteneciente a un tema de fantasía épica. Para ello, se viene utilizando una interfaz gráfica cada vez más vistosa para utilizar un sofisticado inventario de poderes humanos y sobrenaturales (que el jugador desarrolla poco a poco con práctica y muchas horas de juego), recursos monetarios y objetos diversos en propiedad (comprados o encontrados de manera fortuita), para el logro de las metas.

Ejemplos de estos videojuegos son: Final Fantasy, Dragon Quest, Bravely Default, Golden Sun, Chrono Trigger, The Legend of Zelda, Tales of…

2.2.1. Géneros dentro de los videojuegos RPG

En este apartado se pueden ver las distintas variantes que tienen los videojuegos rpg en el mercado actual.

2.2.1.1. JRPG

JRPG es como se conoce a los juegos de rol japoneses. Suele llamárseles de esta forma para diferenciarlos de los RPG occidentales.

Las diferencias entre un JRPG y un RPG occidental suelen ser:

- Los JRPG por lo general tienen unos gráficos estilo “anime” (al menos así era en sus orígenes, luego con el paso del tiempo se optaron por gráficos más realistas) mientras que por lo general, los RPG occidentales suelen caracterizarse por tener un mundo amplio y tramas más oscuras.
- Mientras que los JRPG tienen una progresión más lineal haciendo que tanto la historia como los personajes que en ella aparecen tengan una mayor profundidad. Por cambio, los occidentales, tienen historias más ramificadas que vienen dadas por las diferentes elecciones que hace el jugador a lo largo de la historia.
- Mientras que los RPG occidentales tienen unos combates en tiempo real, los JRPG están caracterizados por los combates por turnos de forma general.

2.2.1.2. Action RPG

Los Action RPG son videojuegos donde la acción y los combates en tiempo real tienen una mayor importancia.

Así pues, los combates son bastante frenéticos ya que hay acción constante. Y el jugador debe reaccionar a todo lo que ocurre mediante sus reflejos de una forma rápida para poder acabar con todos los enemigos.

2.2.1.3. Tactic o Strategic RPG

Este subgénero se caracteriza por tener varias unidades dispuestas sobre el terreno de batalla donde, mediante turnos, el jugador las irá moviendo mediante la rejilla (que es la característica que más distingue este tipo de juegos) para poder derrotar a la IA rival.

2.2.1.4. MMORPG

MMORPG son las siglas correspondientes a “Masive Multiplayer Online Role Playing Game”. Este tipo de juegos conectan a grandes cantidades de jugadores a la vez para que jueguen entre ellos. Ya sea o bien avanzando en la historia mediante mazmorras derrotando enemigos o bien para que peleen entre ellos.

2.3. Motores de videojuegos

Actualmente existe una gran variedad de motores de videojuegos, tanto gratuitos como de pago, de código abierto y cerrado e incluso motores propios de las empresas que solo son utilizados en su ambiente cerrado a la hora de desarrollar sus videojuegos.

2.3.1. Unreal Engine

Unreal Engine es un motor de videojuegos programado en C++ y desarrollado por la compañía Epic Games (desarrolladores de juegos como Unreal Tournament y Gears of War). Actualmente en su versión 4, es un motor totalmente gratuito donde sólo hay que aportarles un 5% de los beneficios obtenidos con la comercialización del videojuego.

A nivel gráfico, es compatible con OpenGL, Vulkan y DirectX 11 y 12, lo que hace que pueda exportar a distintas plataformas como PC (Windows y Linux), MAC, Web (HTML5), dispositivos móviles (Android e iOS) y la mayoría de las consolas actuales (PS4, Xbox One y Nintendo Switch) además de dar soporte a VR.

Gracias a su gran potencia gráfica, también es usado fuera del ámbito de los videojuegos, como por ejemplo para realizar cinemáticas de animación.
En cuanto a funcionalidades, Unreal permite el scripting en C++ (previo desuso de su propio lenguaje UnrealScript similar a Java) lo cual apoya el sistema de Blueprints (bloques de código encapsulados que definen acciones y comportamientos) que permite hacer videojuegos sin tener muchos conocimientos sobre scripting. También incluye todo tipo de editores: texturas, mapas, terrenos, partículas…
Por último, Unreal Engine tiene una amplia comunidad que se apoya mutuamente mediante la compartición de assets, plug-in, etc.

2.3.2. GameMaker: Studio

GameMaker:Studio es una herramienta escrita en Delphi y un kit de desarrollo (SDK) pensada para la creación de videojuegos. Software gratuito (existe una versión comercial de pago con mayores características), está pensado para aquellos desarrolladores novatos que tienen un conocimiento menor de programación. Aunque aquellos más experimentados, mediante el lenguaje de scripting propio del motor, Game Maker Language(GML), pueden personalizar aún más sus juegos y expandir enormemente sus características.
Los juegos creados con GameMaker:Studio pueden ser distribuidos bajo cualquier licencia sujeta a los términos del EULA de GameMaker y para cualquiera de las siguientes plataformas: PC (Windows), móviles(Andriod) y Web (HTML5).
La interfaz de GameMaker, al estar pensada para aquellos usuarios no familiarizados con el desarrollo de videojuegos, funciona con el método “drag&drop”. Siendo así muy intuitiva ya que aprovechando las bibliotecas de acciones que trae por defecto el motor (movimiento, dibujo básico…) simplemente arrastrando dichas acciones al marco de trabajo.
Como se ha mencionado anteriormente, GameMaker utiliza su propio lenguaje. Y aunque el motor esté pensado para realizar videojuegos en 2D, mediante el uso de este lenguaje, se pueden lograr grandes resultados en videojuegos 3D.

2.3.3. Cry Engine

Cry Engine es un motor de videojuegos desarrollado por la compañía Crytek e introducido por primera vez en su videojuego Far Cry. El motor, al igual que Unreal Engine, tiene una potencia gráfica muy grande y es por eso que los resultados son muy vistosos. Está preparado para exportar a varias plataformas como PC, PS4, Xbox One y VR.

El motor es totalmente gratuito, sin ninguna obligación de aportar a Crytek ningún porcentaje de los beneficios obtenidos.
Cuenta con una gran comunidad y un amplio mercado de assets, pero en contraparte cuenta con una amplia curva de dificultad.

2.3.4. Xenko Engine

Xenko Engine es un motor 2D y 3D de código abierto desarrollado en 2015 por la compañía nipona Silicon Studio (desarrolladores de varios videojuegos como la saga Bravely) capaz de exportar a múltiples plataformas (PC, iOS, Android, VR y Xbox One, con las siguientes versiones del motor vendrá la exportación al resto de consolas).

El software es gratuito, pero al igual que con motores como Unity o Unreal, tiene versiones en función de para que se usen (para desarrollos indie, para estudios profesionales, para estudios profesionales que buscan tener más flexibilidad y crear sus herramientas dentro del motor o una configuración del motor totalmente personalizada).
A nivel de funcionalidades es un motor completo de altísimo nivel orientado a componentes escrito en lenguaje C# (soporta la última versión del lenguaje, 7.0), que ofrece unos resultados gráficos altísimos (compatibilidad tanto como con DirectX12 y Vulkan).
La herramienta principal dentro de Xenko es el llamado Game Studio que permite importar los assets, crear y organizar las escenas, crear los scripts, compilar y ejecutar los juegos. Además de todo esto viene equipado con varios editores: partículas, sprites, rendering, UI… también contiene el famoso sistema de prefabs (objetos de la escena creados en el editor que se guardan como assets).

2.3.5. Motores de videojuegos RPG, RPGMAKER.

Tanta es la fama del género rpg que incluso se han desarrollado motores para crear videojuegos de éste género.
Ejemplo de ello es el motor RPG Maker, el cual cuenta con un gran número de revisiones y versiones que han ido saliendo con el paso de los años (desde 1988 hasta el día de hoy para ser exacto).
RPG Maker proporciona grandes facilidades al desarrollador tales como un editor de mapas, un editor de eventos y un editor de base de datos. A su vez, también incluye paquetes de assets como gráficos para mapeados, personajes, música, efectos de sonido, etc.

En su última versión RPG Maker MV aunque trae algunas funcionalidades nuevas como los nuevos “pluggins” para ayudar a los desarrolladores que no tengan muchos conocimientos de programación y algunas expansiones de tamaño de algunas bases de datos (como la de objetos que pasa de 999 a 2000 objetos) tiene algunas limitaciones o desventajas que cabe mencionar. Para empezar es de pago (73,99€) y actualmente cuenta con un modelo de negocio basado en “dlc”, es decir, expansiones de contenido descargable (nuevos assets, etc) también de pago. Pese haber ido avanzando de versión, el motor en sí no ha sido actualizado en mucho tiempo, es decir, sigue siendo lo mismo de siempre, aunque ofreciendo mejor resultado. Puede parecer que es poca cosa, pero al mirar en precio es algo a tener en cuenta puesto que la versión anterior RPG Maker VX Ace, no cambia mucho en cuanto a funcionalidad pero si en precio, ya que es mucho más barato.
Además de eso, se arrastran problemas de versiones anteriores, por ejemplo: si se desea cambiar una plantilla de las que ofrecer el motor, se debe hacer siempre desde la carpeta del programa.

Figura 1- RPGMaker en Steam

Figura 2- Expansiones RPGMaker en Steam

2.3.6 Comparativa entre motores

Este apartado aporta una tabla comparativa de los motores vistos en los apartados anteriores.

	Unreal Engine	GameMaker	CryEngine	Xenko
Soporta 2D	Si	Si	Si	SI
Soporta 3D	Si	Si (con scripting)	Si	Si
Lenguaje de scripting	C++	GameMaker Languaje (GML)	C++	C#
Curva de aprendizaje	Alta	Baja	Alta	Media
Tienda de assets	Si	Si	Si	Si
Plataformas exportables	- Windows PC - Playstation 4 - Xbox One - MacOS X - iOS - Android - VR - Linux - SteamOS - HTML5.	- Windows desktop - Mac OS X - Ubuntu - Android - iOS - FireTV - Android TV - Microsoft UWP - HTML5 - PlayStation 4 - Xbox One.	- Oculus Rift - Windows PC - Linux PC - Xbox One - Playstation 4	- Windows - Android - iOS - VR - Xbox One - Microsoft (UWP)
Gratuito	Si	No	Si	Si

Tabla 1- Comparativa entre motores

2.4 Referencias

En este apartado se muestran las referencias que se han tomado para realizar el videojuego.

2.4.1. Bravely Default y Bravely Second

Figura 3- Portada Bravely Default

Figura 4- Portada Bravely Second

Bravely Default y Bravely Second son JRPG desarrollados por la compañía Silicon Studio y publicados por Square Enix (en Japón) y Nintendo (en el resto del mundo) en 2012 y 2015 respectivamente. Ambos juegos forman parte de la familia de 3DS. Estos juegos han sido aclamados por el público, puesto que supone una vuelta a los antiguos JRPG con los elementos clásicos del género: historia que atrapa, personajes muy carismáticos, combates por turnos y el motivo por el cual es una de las principales referencias de este trabajo, el sistema de trabajos (presente también en otros juegos como Final Fantasy III).

2.4.2. Final Fantasy VI y VII

Figura 5- Portada Final Fantasy VI

Figura 6- Portada Final Fantasy VII

La saga Final Fantasy es famosa en todo el mundo con una gran trayectoria a lo largo de los años, y nos ha dejado grandes títulos como Final Fantasy VI y Final Fantasy VII. Juegos desarrollados y publicados por SquareSoft/Square.Co, conocida actualmente como Square Enix.

Estos dos juegos han tenido un gran impacto en la historia de los videojuegos por la narrativa de sus historias, en concreto el 7 que supuso toda una revolución en los videojuegos cuando salió al mercado.

Si bien es verdad que los JRPG son característicos por los combates por turnos, hay que decir que estos juegos tienen un componente que hace los turnos algo más dinámicos, y este es el ATB (Action

Time Battle) que consiste en tener barras de tiempo que determinan cuando el jugador o la unidad enemiga puede hacer una acción en función de si dicha barra está llena o no.

Figura 7- Ejemplo de ATB en Final Fantasy VI

2.4.3. Digimon World DS

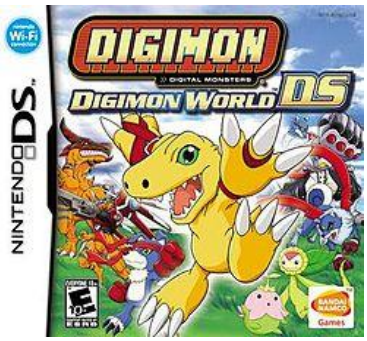


Figura 8- Portada Digimon World DS

Digimon World DS es un videojuego de rol desarrollado por BEC y distribuido por Bandai Namco Games.

Este videojuego es bastante parejo a la saga RPG Pokémon, puesto que se basa en coleccionar y entrenar digimon. A la hora de batalla, es su interfaz lo que se toma como referencia para este juego, ya que solo se ven los enemigos en la parte superior de la pantalla mientras que en la inferior solo se ven los retratos de los digimon empleados con sus datos de vitalidad y maná.

Figura 9- Ejemplo de Batalla en Digimon World DS

2.4.4. Suikoden 2

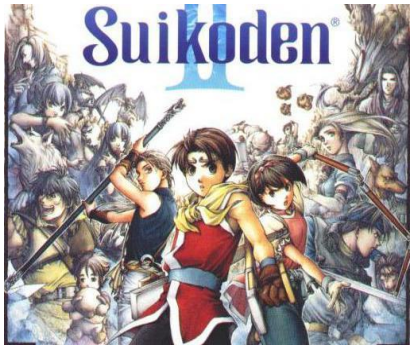


Figura 10- Portada Suikoden 2

Desarrollado y distribuido por Konami, Suikoden 2 es un JRPG para la primera Playstation (PSX) donde disponemos de todos los elementos propios del género, con la nota de que hay una gran cantidad de personajes obtenibles (108 por los 5,6 que solemos encontrar en la mayoría de JRPG). Y aunque el motivo por el cual es una referencia no es visible a simple vista, gracias a las distintas wikis, se puede ver como es el crecimiento de estadísticas de los personajes.

2.5. Conclusión del género

Como se ha podido observar, el RPG es un género a la orden del día que desde el momento de su nacimiento no ha parado de evolucionar y de generar cada vez más público.

Primeramente fueron juegos con gráficos 2D por todas las plataformas, o casi todas, ya que desde tiempos de la Super Nintendo hay juegos de esta índole.

Seguidamente, pese que al principio de las consolas como PlayStation hubo muchos juegos en 2D, se dio el salto al 3D con juegos icónicos como Final Fantasy VII o Chrono Cross, entre muchísimos otros títulos. Y desde entonces ya se entró plenamente en la época del 3D y casi todos los RPG llegaron al mercado en este formato.

A nivel mecánico ha habido bastante evolución desde los primeros turnos básicos (ej. Pokémon), pasando por turnos con barra de tiempo individual (Final Fantasy VI, Final Fantasy VII), turnos con barra de tiempo colectiva (Grandia), combate en tiempo real (Final Fantasy XV),…

Y la evolución no se limita únicamente a nivel mecánico, puesto que la narrativa de los juegos también ha avanzado considerablemente. Pues con la aparición de los RPG occidentales y su sistema de elecciones y diálogos se abrió un nuevo horizonte capaz de hacer más interesante la forma de ver el juego como tal, al punto de verlo como un libro interactivo (la saga The Witcher, Fallout o Mass Effect son ejemplos de narrativa moderna).

Videojuegos como la saga Persona, a partir de su tercera entrega supusieron una evolución dentro del género, aparte de ser un videojuego RPG, introdujeron un componente social, es decir, un simulador de vida donde el jugador debe hacer la vida de un adolescente (estudiar, ejercitarse, relacionarse…).

En definitiva, el RPG es un género muy grande y expandido a lo largo del mundo a través de todas sus variantes. Y a día de hoy generan una gran cantidad de expectación entre el público lo cual puede verse en los éxitos recientes de juegos como Persona 5, The Witcher 3, Zelda Breath of the Wild y Nier: Automata.

3. Objetivos y justificación

El objetivo de este proyecto es la creación de un videojuego en 2D incluido dentro del género RPG, más concretamente en su variante japonesa, es decir, JRPG.

El videojuego se construirá con el motor comercial Unity 5. Para ello se implementará todo un sistema de clases genérico para hacer este tipo de videojuegos. Este sistema tendrá soporte para varias funcionalidades para videojuegos jrpg entre las que se destacan: controlador del juego y batalla, sistema de trabajos, sistema de (crecimiento de) estadísticas, bibliotecas de objetos, enemigos, objetivos (misiones) y habilidades.

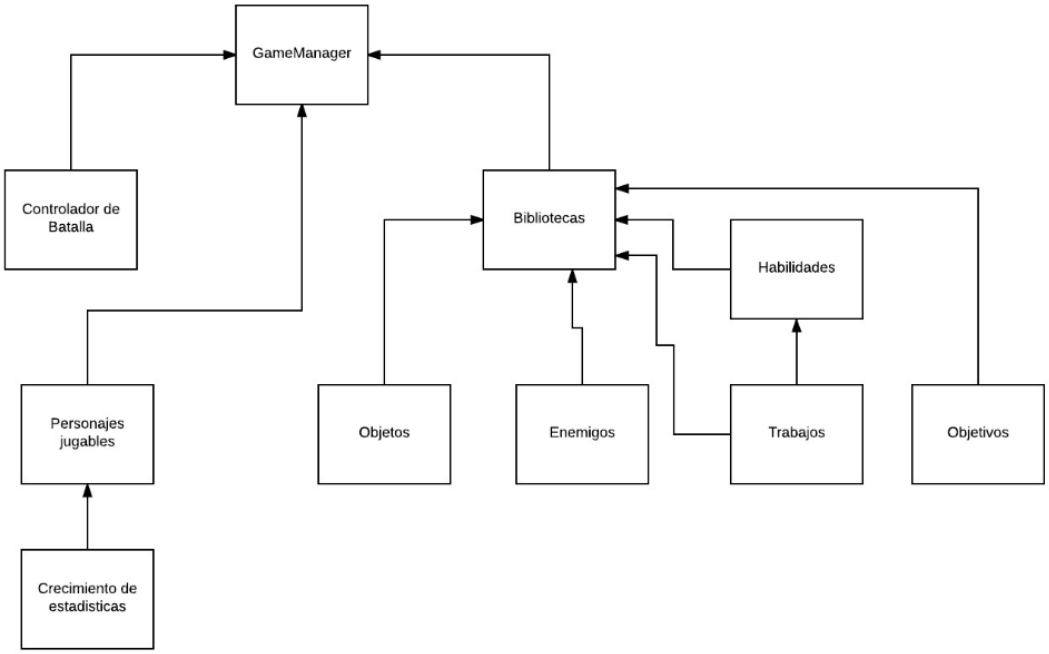


Figura 11- Funcionalidades del sistema de clases

3.1. ¿Por qué Unity 5?

Se utilizará el motor Unity 5 para la construcción del videojuego porque es una herramienta potente, comercial y tiene versión gratuita. Pero el motivo más importante del porque usar Unity 5 es para aprender a utilizarlo junto con todas sus funcionalidades ya que es una herramienta de desarrollo muy asentada en el mercado.

Unity ofrece una gran cantidad de ventajas para el desarrollador que van desde la fácil implementación de prototipos hasta la exportación a 21 plataformas diferentes con un solo código, lo cual hace que tenga gran fuerza en el mercado. Por otro lado, la curva de aprendizaje es baja lo cual hace que sea rápido en comprender como funciona tanto el motor como la estructura basada en componentes. Además, para complementar dicha curva de aprendizaje, la comunidad de Unity es enorme y muy activa, al igual que la documentación, que se extiende más allá de los foros oficiales hasta llegar a Youtube donde hay una gran cantidad de tutoriales para llevar

a cabo cualquier idea. Además de tener una tienda de assets que no para de actualizarse día a día con las aportaciones de la comunidad.

A nivel de scripting, el motor permite hacer scripts en dos lenguajes de programación diferentes: C# y JavaScript, donde además son combinables dentro del mismo código.

A pesar de que no puede competir a nivel gráfico con motores como Unreal o Cry Engine, tras la introducción de la versión 5, Unity dio un salto de calidad en cuanto a potencia y se abrieron todo un mundo de posibilidades.

Dicho esto, lo que se pretende es crear una base para hacer videojuegos RPG en Unity 5.

3.2. ¿Por qué un j-rpg?

Actualmente el JRPG es un género que está presente independientemente del paso de los años. Es un género bastante aclamado y que tiene un público bastante fiel tanto en su variante más clásica (juegos más estratégicos con turnos) como en su variable más moderna (juegos con más acción, rapidez y escenas cinematográficas).

Prueba de esto son los distintos lanzamientos que tenemos hoy en día, como pueden ser:

- Bravely Default (Nintendo 3DS, año 2012)
- Bravely Second (Nintendo 3DS, año 2015)
- Persona 5 (PS3, PS4, año 2016)
- Tales of Berseria (PS3, PS4, PC, año 2017)
- Tales of Zestiria (PS3,PC, año 2016)
- Final Fantasy 15 (PS4, Xbox One, año 2016)
- Pokémon Sol y Luna (Nintendo 3DS, año 2016)
- Pokémon UltraSol y UltraLuna (Nintendo 3DS, año 2017)
- Dragon Quest XI (Nintendo 3DS, año 2017)
- Dragon Quest VII (Nintendo 3DS, año 2016)
- Dragon Quest VIII (Nintendo 3DS, año 2017)
- Nier: Automata (PS4, PC, año 2017)
- Xenoblade Chronicles (Wii, año 2012, New Nintendo 3DS, año 2015)
- Xenoblade Chronicles X (WiiU)
- Fire Emblem Fates (Nintendo 3DS, año 2015)
- Fire Emblem Echoes: Shadows of Velentia (Nintendo 3DS, año 2017)

4. Metodología

En este apartado se expone la metodología aplicada en el desarrollo del proyecto.

4.1. Metodología de desarrollo iterativo/evolutivo

La metodología utilizada para este proyecto ha sido una metodología basada en iteraciones, donde en cada iteración se ha presentado un prototipo que ha permitido que el proyecto haya ido evolucionando con cada iteración al añadir cada vez más funcionalidades.

Las iteraciones se han producido en intervalos de tiempo de 2 semanas cada una.

4.2. Control de versiones

Para el desarrollo del proyecto se ha escogido un repositorio en Github para tener un control de versiones.
De ésta forma se puede elegir que salvar con el fichero gitignore y tener una copia de seguridad en la red de todos los avances realizados. Cada copia de seguridad se ha hecho cada 2 semanas coincidiendo con la iteración de la metodología.

5. Cuerpo del trabajo

En este apartado se expone el desarrollo del proyecto dividido en dos partes:

-Primero se encuentra el documento de diseño del juego (GDD), donde se muestran todos los aspectos de diseño a tener en cuenta a la hora de crear un videojuego.

-La siguiente y última sección está dedicada al desarrollo e implementación del videojuego. Donde se explican entre otras cosas, las herramientas utilizadas y cómo funcionan.

5.1. Documento de diseño del juego (GDD)

En este bloque se puede encontrar todo el diseño teórico del videojuego a nivel de mecánicas, niveles, arte e historia.

5.1.1. Información sobre el juego

Ivankar's Tale es un título j-rpg de corte clásico. Con esto dicho, el juego consistirá en explorar el mapa, hacer progresos en la historia (y con los personajes) mediante exploración y una mecánica de batallas por turnos con barra de tiempo (Action Time Battle).

5.1.2. Jugabilidad y mecánicas

5.1.2.1. Exploración

La exploración se divide en 3 partes: Mapamundi, ciudades y mazmorras.

- Mapamundi: es el mundo entero, por el cual tendremos que movernos para acceder a mazmorras o a ciudades.

- Ciudades: en las ciudades tendremos NPC's con los cuales podremos interactuar ya sea o bien para obtener información (importante o no) o bien para obtener objetos. A su vez dentro de la ciudad podremos entrar en los distintos edificios que hay en ellas (casas, tabernas, tiendas,…).

- Mazmorras: están fuera de las ciudades y se accede a ellas (normalmente) desde el mapamundi. En ella podremos encontrar desde enemigos, objetos dentro de cofres hasta puzles para poder avanzar por la mazmorra.

5.1.2.2. Batalla

La batalla está compuesta de los 3 primeros integrantes del grupo y de 3 enemigos propios de la zona del mapa, los cuales son elegidos de forma aleatoria.

Ésta se produce de forma aleatoria, es decir, la batalla puede empezar en cualquier momento mientras se está caminando o bien por el mapamundi o bien por alguna mazmorra.

5.1.2.2.1. Action Time Battle

Este es el sistema que define cómo es la batalla. La principal característica de este sistema es la **barra de tiempo**. Cada unidad dentro de la batalla dispone de una de éstas y va llenándose conforme avanza el tiempo. La velocidad con la que se van llenando estas barras depende de la estadística de velocidad de cada unidad (a mayor velocidad, más rápido se llenará la barra). Una vez está completa la barra,

el tiempo de la batalla se detiene y se elige la acción que se va a realizar la unidad (las controlables por el jugador y las enemigas por su inteligencia artificial).

5.1.2.2.2 Sistema de daño

El sistema de daño en la batalla diferencia entre si se está infringiendo daño mágico o daño físico, y en función del tipo aplica una de las siguientes formulas:

Formula de daño físico (en habilidades): *fuerzaAtacante/defensaObjetivo x dañoHabilidad*.

Formula de daño físico (en ataques normales): *fuerzaAtacante – defensaObjetivo*.

Formula de daño mágico: *intelectoAtacante/defensaMagicaObjectivo * dañoHabilidad*.

A su vez, a la hora de aplicar curaciones mediante habilidades, se aplica la siguiente formula:

Formula curación (habilidades): *potenciaHabilidad+intelectoCurador*0,8*.

5.1.2.2.3 Acciones durante la batalla

- Atacar: El personaje realiza un ataque físico normal al objetivo seleccionado.
- Habilidades: Muestra una lista de habilidades en función del trabajo que tiene asociado el personaje para después seleccionar tanto la habilidad deseada como el objetivo de esta.
- Objetos: Muestra una lista con todos los objetos usables del inventario (pociones) para después seleccionarlos tanto al objeto como al objetivo.
- Huir: Produce el escape del grupo mediante una probabilidad del 50%.

5.1.2.3 Sistema de trabajos

Los trabajos definen en cierta forma el rol que tienen los personajes dentro del grupo aportando características únicas a cada uno. Aportando desde una modificación de estadísticas (aumentando las más favorables y bajando las que menos) mediante multiplicadores, a una lista de habilidades propias de cada clase (p.ej: Mago Blanco: Cura (cura 30 puntos de vida al objetivo)).

Los trabajos, al igual que los personajes, también pueden subir de nivel desbloqueando así nuevas habilidades. El nivel de cada trabajo es individual de cada personaje, es decir, todos los personajes pueden llevar asociado del mismo trabajo pero no necesariamente éste tiene que tener el mismo nivel (salvo que esté al nivel máximo).

Los trabajos se componen de 10 niveles, y para alcanzar el máximo nivel deben de adquirir experiencia al igual que los personajes. El criterio para subir de nivel es el siguiente:

Nivel	Experiencia necesaria para subir de nivel
1	0
2	30
3	100
4	300
5	400
6	800
7	1500
8	2500
9	5000
10	8000

Tabla 2- Niveles de los trabajos

5.1.2.3.1 Trabajos y habilidades

5.1.2.3.1.1 Mago Negro

Grandes hechiceros, derrotan a sus enemigos mediante fuertes magias ofensivas, las cuales cubren distintos elementos como por ejemplo: fuego, hielo y rayo.

Estadística	Modificadores
Vida	0
Maná	0.3
Fuerza	-0.4
Defensa	-0.2
Intelecto	0.4
Defensa mágica	0.3
Velocidad	0

Tabla 3- Modificadores Mago Negro

Nivel de Trabajo	Habilidades desbloqueadas
1	Piro
2	Frio
3	Rayo
4	Piro+
5	Frio+
6	Rayo+
7	Gravedad
8	Bio
9	Piro++,Frio++,Rayo++
10	Fulgor

Tabla 4- Habilidades del Mago Negro por nivel

5.1.2.3.1.2 Mago Blanco

Los magos blancos están más enfocados al apoyo del grupo mediante magias curativas, aunque al tener afinidad con la luz pueden ejercer daño mediante hechizos de luz.

Estadística	Modificadores
Vida	0
Maná	0.3
Fuerza	-0.3
Defensa	-0,4
Intelecto	0.2
Defensa mágica	0.4
Velocidad	0

Tabla 5- Modificadores Mago Blanco

Nivel	Habilidades desbloqueadas
1	Cura
2	Aero
3	Esna
4	Aero+
5	Cura+
6	Lázaro
7	Aero++
8	Cura++
9	Lázaro+
10	Sanctus

Tabla 6- Habilidades del Mago Blanco por nivel

5.1.2.3.1.3 Monje

Grandes guerreros espirituales, han dejado atrás toda debilidad. Prueba de ello es su gran poder ofensivo sin reserva alguna. Se caracterizan por utilizar su energía espiritual para ejercer un gran daño a los enemigos.

Estadística	Modificadores
Vida	0.3
Maná	-0.3
Fuerza	0.4
Defensa	0.2
Intelecto	-0.4
Defensa mágica	-0.2
Velocidad	0.2

Tabla 7- Modificadores Monje

Nivel	Habilidades desbloqueadas
1	Golpe fuerte
2	Golpe de chakra
3	Dragón interior
4	Curación espiritual
5	Palma de aire
6	Puntos de presión
7	Concentración
8	Onda de choque
9	Dragón rojo
10	Liberación interior

Tabla 8- Habilidades del Monje por nivel

5.1.2.3.1.3 Caballero

Espada y escudo es el rasgo más característico de los caballeros. Se centran en ofrecer protección a sus compañeros haciendo gran uso de su escudo. Aunque se centran mayoritariamente en la defensa, también pueden aportar un buen poder ofensivo.

Estadística	Modificadores
Vida	0.2
Maná	-0.2
Fuerza	0.2
Defensa	0.4
Intelecto	-0.3
Defensa mágica	0
Velocidad	-0.2

Tabla 9- Modificadores Caballero

Nivel	Habilidades desbloqueadas
1	Golpe de escudo
2	Arremetida
3	Defensa férrea
4	Golpe de caballero
5	Valentía
6	Proteger
7	Escudo aturdidor
8	Golpe de valor
9	Protección máxima
10	Súper Arremetida

Tabla 10- Habilidades por nivel del Caballero

5.1.2.4 Sistema de diálogos

A lo largo del juego nos encontraremos con npc con los que podremos interactuar mediante diálogo. Estos diálogos pueden desencadenar (o no) cambios en el mapa (p.ej: se desbloquea el acceso a una mazmorra tras hablar con un npc), proporcionarnos información que necesitamos o incluso objetos.

5.1.2.5 Objetos

A lo largo del juego encontraremos distintos objetos tales como: pociones, armas, armaduras…

5.1.2.5.1 Pociones

Las pociones permitirán recuperar la vida o el maná de los personajes.

Poción	Recupera 50 puntos de vida	30g
Superpoción	Recupera 150 puntos de vida	50g
Ultrapoción	Recupera 500 puntos de vida	150g
MAX. Poción	Recupera toda la vida	200g
Éter	Recupera 50 puntos de maná	30g
Éter MAX.	Recupera 150 puntos de maná	60g
Elixir	Recupera toda la vida y maná	500g

Tabla 11- Pociones disponibles

5.1.2.5.2 Equipables

Los objetos equipables proporcionan un aumento de estadísticas del personaje al cual se equipan. Este aumento puede ser en una o en varias estadísticas.

5.1.2.5.2.1. Armas

5.1.2.5.2.1.1. Espadas

Espada de cobre	Vida: 0 Mana: 0 Fuerza:5 Defensa: 0 Intelecto: 0 Defensa mágica: 0 Velocidad: 0
Espada de plata	Vida: 0 Mana: 0 Fuerza:8 Defensa: 0 Intelecto: 0 Defensa mágica: 0 Velocidad: 0
Espada de oro	Vida: 0 Mana: 0 Fuerza:10 Defensa: 0 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 12- Espadas disponibles

5.1.2.5.2.1.2. Hachas

Hacha de piedra	Vida: 0 Mana: 0 Fuerza:8 Defensa: 0 Intelecto: 0 Defensa mágica: 0 Velocidad: 0
Hacha de cuarzo	Vida: 0 Mana: 0 Fuerza:10 Defensa: 0 Intelecto: 0 Defensa mágica: 0 Velocidad: 0
Hacha de hierro	Vida:5 Mana: 0 Fuerza:13 Defensa: 0 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 13- Hachas disponibles

5.1.2.5.2.1.3. Bastones

Bastón desgastado	Vida: 0 Mana: 0 Fuerza: 0 Defensa: 0 Intelecto:5 Defensa mágica: 0 Velocidad: 0
Bastón de roble	Vida: 0 Mana: 0 Fuerza: 0 Defensa: 0 Intelecto:8 Defensa mágica: 0 Velocidad: 0
Bastón de metal	Vida: 0 Mana: 0 Fuerza: 0 Defensa: 0 Intelecto:10 Defensa mágica: 0 Velocidad: 0

Tabla 14- Bastones disponibles

5.1.2.5.2.2. Armaduras

5.1.2.5.2.2.1. Cabeza

Sombrero de aprendiz	Vida:10 Mana:5 Fuerza: 0 Defensa: 0 Intelecto:5 Defensa magica:8 Velocidad: 0
Casco de cuero	Vida:15 Mana: 0 Fuerza:5 Defensa:5 Intelecto: 0 Defensa mágica: 0 Velocidad: 0
Yelmo de hierro	Vida:30 Mana: 0 Fuerza:5 Defensa:8 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 15- Armaduras para la cabeza disponibles

5.1.2.5.2.2.2. Pecho

Toga de aprendiz	Vida:10 Mana:10 Fuerza: 0 Defensa:2 Intelecto:5 Defensa mágica: 3 Velocidad: 0
Pechera de cuero	Vida:15 Mana: 0 Fuerza:5 Defensa:5 Intelecto: 0 Defensa mágica: 0 Velocidad:2
Peto de hierro	Vida:35 Mana: 0 Fuerza:7 Defensa:10 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 16- Armaduras para el pecho disponibles

5.1.2.5.2.2.3. Manos

Guantes de aprendiz	Vida:5 Mana:5 Fuerza: 0 Defensa:2 Intelecto:5 Defensa mágica: 4 Velocidad: 0
Guantes de cuero	Vida:15 Mana: 0 Fuerza:5 Defensa:3 Intelecto: 0 Defensa mágica: 0 Velocidad:2
Guanteletes de hierro	Vida:30 Mana: 0 Fuerza:7 Defensa:10 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 17- Armaduras para las manos disponibles

5.1.2.5.2.2.4. Piernas

Pantalones de aprendiz	Vida:10 Mana:5 Fuerza: 0 Defensa:3 Intelecto:6 Defensa mágica: 4 Velocidad: 0
Pantalones de cuero	Vida:20 Mana: 0 Fuerza:6 Defensa:5 Intelecto: 0 Defensa mágica: 0 Velocidad:3
Quijotes de hierro	Vida:30 Mana: 0 Fuerza:7 Defensa:10 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 18- Armaduras para las piernas disponibles

5.1.2.5.2.2.5. Pies

Sandalias de aprendiz	Vida:6 Mana: 0 Fuerza: 0 Defensa:3 Intelecto:5 Defensa mágica: 4 Velocidad:3
Botas de cuero	Vida:50 Mana: 0 Fuerza:5 Defensa:8 Intelecto: 0 Defensa mágica: 0 Velocidad: 0
Grebas de hierro	Vida:50 Mana: 0 Fuerza:5 Defensa:8 Intelecto: 0 Defensa mágica: 0 Velocidad: 0

Tabla 19- Armaduras para los pies disponibles

5.1.2.5.3 Obtención de objetos

A lo largo del juego, el jugador se encontrará con muchos objetos. Y estos serán obtenibles de cofres en mazmorras, comprándolos a los vendedores repartidos por las distintas ciudades, como botín enemigo al terminar una batalla o bien a partir de realizar alguna conversación con un npc.

5.1.3. Historia, características y personajes

5.1.3.1. Historia

En este apartado se describe la ambientación e historia del juego, se explicará en 2 partes. La primera, el marco general que relata el contexto a gran escala dentro del universo propuesto y, la segunda, el marco actual que relata la situación que se vive en el juego al empezar a jugar.

5.1.3.1.1. Marco general

Nuestra historia debe comenzar con el nacimiento de una nación. El vasto terreno que domina esta nación antaño fue gobernado por diez clanes de diferentes razas que mantenían culturas distintas y en algunos casos fueron rivales durante años.

Con la aparición de nuevas tecnologías basadas en la energía libre fue necesaria la extracción en grandes cantidades de minerales especialmente conductores, tales como: el Berilio para la conducción, cristales mágicos para la contención, el azufre para las mezclas químicas y los metales pesados para la construcción mecánica. Debido a estas nuevas necesidades creó pactos y estableció amistades entre algunos de los clanes más avanzados.

La unión de los cuatro clanes más grandes obligó a los demás integrantes del territorio a tomar medidas para equilibrar los fuertes canales de capital y poder que movían dichos clanes. Llegando al extremo de ser necesaria la protección militar de las caravanas de comerciantes para evitar el boicot.

Entre todo este caos, cuando las alianzas y los pactos temblaban por culpa de los repetitivos ataques surgió un líder. Alguien capaz de estabilizar el senado con su palabra y capaz de dominar a las otras 6 naciones con la fuerza y la negociación. Así comenzó la Guerra Civil.

Aunque se desataron terribles batallas lo cierto era que jamás se había visto en el territorio un avance de las tecnologías, la calidad de vida y la seguridad de tal envergadura. Dicho Líder, Julius, fue votado como gobernador por las clases selectas que veían en él el futuro de una gran nación.

Más tarde cuando las demás tribus aceptaron la rendición, o la unión al conjunto, dicho líder se autoproclamó Emperador y a los diez clanes los denominó Novum Imperium. El comienzo de este sistema asustó a muchos pero las nuevas legislaciones no dejaron de ser más que un título honorífico.

Se mantuvo el senado, las rutas comerciales y la mayoría de los impuestos no se modificaron, incluso aparecieron nuevas formas religiosas y en cierto modo hubo libertad de expresión más allá de las restricciones tradicionalistas. La gente aprendió a convivir y por primera vez las distintas razas viajaron entre territorios, conocieron distintas costumbres, aprendieron a empatizar con los demás.

Sin embargo, hubo un clan que se negó a cooperar como todos los demás. Su tradición siempre había sido renegada y ermitaña, sus costumbres no eran aceptadas por todos aun cuando antaño fueron los oráculos y los sacerdotes mejor recibidos y más afamados de todo Novum Imperium. Siempre habían sido poderosos y arrogantes demostrando su poder en muchas ocasiones contra los clanes más cercanos. En el pasado, ellos mismos habían protegido de maldiciones y enfermedades a todos los clanes por igual a cambio de su superioridad religiosa.

Ninguna otra religión había crecido como la suya en todo el territorio, pero ahora…

La octava Tribu jamás había entendido, como las demás, la supremacía de Julius y sus trifulcas y desinterés por el imperio había crecido desde la guerra civil. Esta tribu era tradicionalista y los oráculos habían hablado. Julius no debía gobernar.

Nunca había sido una aldea comercial, su principal inversión siempre había sido la protección y la magia de sangre, magia moralmente prohibida en el Novum Imperium.

Hubo varios atentados contra el emperador. Intentos de asesinatos que no lograron más que unir toda la rabia de los 9 clanes contra ellos.

Julius era un líder sensato y por ello intentó parlamentar, pero algunos de los mensajeros jamás regresaban. Ante todo Julius siempre había sido conocido por ser un estratega frío y preciso, nunca le importó nada más que su querido imperio y por ello estaba dispuesto a erradicar de la tierra a todos esos líderes religiosos que atentaban contra su palabra.

Llegó a interceptar mensajes provenientes de territorios exteriores al imperio con planes muy específicos para el Regicidio y la masacre, por medio de rituales y magias del pasado, de todos los demás clanes.

Si era guerra lo que deseaban, guerra tendrían. Dijo Julius.

Sin embargo el senado le recomendó una alternativa, el exilio.

Era cierto que su unificación había llamado la atención de grandes naciones exteriores y ahora los problemas podrían venir de fuera y no de dentro. Librar una segunda guerra civil podría ser una gran oportunidad para acabar con su joven imperio. Por tanto, Julius, aprobó un decreto por el cual ningún componente de la octava tribu podría convivir jamás dentro de su imperio ergo debían de ser exiliados de todas las ciudades del Novum Imperium.

En las ciudades más fieles los propios vecinos se manifestaron con purgas indiscriminadas contra los Zhule. Muchos murieron en las sublevaciones y otros tantos lo hicieron en las selvas y los desiertos a los que fueron expulsados.

Para la mayoría del Imperio habían sido las decisiones acertadas, **pero no todos pensaban como él**, incluso dentro de sus propias ciudades, en sus propias razas para algunos había resultado ser una acción cruel e injustificada. Casi un exterminio y una demostración de fuerza innecesaria.

Consumidos por el odio y la rabia los pocos supervivientes Zhule, expulsados de sus templos sagrados y despojados de todas sus reliquias mágicas que tanto adoraban, se ocultaron en una antigua pirámide situada muy al oeste, escondida entre las montañas.

Allí reactivaron la magia más profunda de las raíces de la tierra y encontraron un secreto que llevaba miles de años guardado, un espejo. Se trataba de un escudo astral que tiempo atrás

sirvió para proteger a las tierras de los 10 clanes del ataque de ancianas serpientes de fuego que ahora dormían famélicas en grandes cavernas bajo tierra. Existía uno en cada una de las seis pirámides que una vez fueron construidas por los Zhule por todo el territorio.

Si encontraban la forma de acabar con el escudo, las mismas serpientes que una vez fueron expulsadas por el pueblo Zhule volverían como demonios vengativos para acabar con sus enemigos...

Así es como los sacerdotes Zhule encuentran la solución a sus problemas, Ivankar, un adolescente superviviente de la masacre consiguió llegar al templo. Su condición de mestizo les repugnaba, pero al mismo tiempo les era útil, pues le mandarían hacer todo el trabajo sucio una vez lo instruyeran en la religión Zhule y de esta forma no correr ningún peligro.

Una vez terminada la formación de Ivankar, él y su amigo Axel parten hacia el imperio para realizar las tareas que les han sido encomendadas por los sacerdotes.

5.1.3.1.2. Contexto actual

Enviado por los sacerdotes Zhule, Ivankar va en búsqueda de uno de los artefactos necesarios para realizar el ritual. En su camino encuentran un pueblo de reciente construcción y deciden entrar para verlo, allí conocen a Leana junto con los problemas de los mineros del pueblo con la mina al este. Ivankar y su grupo deciden ir a investigar y encontrar la respuesta a los misterios que plantean los mineros.

5.1.3.2. Personajes

Los personajes tendrán una serie de estadísticas (vida, fuerza, defensa, defensa mágica, inteligencia, velocidad…) base que irán creciendo cada vez que suban de nivel. A su vez, estas estadísticas se verán modificadas por dos factores: equipo (armas y armaduras que suben o bajan determinadas características) y por los trabajos/profesiones que tenga asociados/as el personaje. Cada trabajo proporcionará consigo un crecimiento o modificación de las estadísticas gracias a unos multiplicadores propios de éste (p.ej: La clase guerrero sube vida y fuerza pero baja inteligencia), y al mismo tiempo proporcionará una serie de habilidades propias de éste.

Los personajes estarán agrupados en una party/grupo el cual será administrado y controlado por el jugador, pudiendo así determinar el orden de salida de los personajes a batalla o incluso cambiando personajes dentro de la batalla.

5.1.3.2.1. Crecimiento de los personajes

Los personajes se fortalecen a medida que suben de nivel y esto se ve reflejado en sus estadísticas. Por lo general éstas aumentarán en mayor medida en niveles más bajos y menor en los niveles más altos.

5.1.3.2.1.1. Estadísticas

Las estadísticas de los personajes se categorizan en rangos (S, A, B, C, D, E) y junto con estos rasgos se define el crecimiento de cada una de éstas.

Las estadísticas son: vida, fuerza, defensa, intelecto, defensa mágica y velocidad.

5.1.3.2.1.1.1. Tabla de rango de estadísticas

La siguiente tabla se corresponde al crecimiento por nivel de las siguientes estadísticas: fuerza, defensa, intelecto, defensa mágica y velocidad.

RANGO	Lvl 02-19	Lvl 20-59	Lvl 60-99
S	4-5 (25%)	2-3 (75%)	0-1 (50%)
A	4-5 (90%)	1-2(90%)	0-1 (50%)
B	3-4 (35%)	2-3(35%)	0-1 (50%)
C	2-3 (40%)	1-2(75%)	0-1 (50%)
D	1-2 (40%)	0-1 (90%)	0-1 (50%)
E	0-1 (90%)	0-1 (60%)	0-1 (50%)

Tabla 20- Tabla crecimiento de estadísticas

5.1.3.2.1.1.2. Tabla de vitalidad y maná

El estado de vitalidad aunque siga el mismo patrón que el resto de estadísticas, la cantidad de puntos que aumenta por nivel es distinto, así queda reflejado en la siguiente tabla.

RANGO	Lvl 02-19	Lvl 20-59	Lvl 60-99
S	23-24	9-10	4-6
A	18-20	8-10	4-6
B	10-12	8-9	4-6
C	9-10	7-9	4-6
D	7-9	5-7	4-6
E	5-7	3-4	4-6

Tabla 21- Tabla de crecimiento de estadísticas vida y maná

5.1.3.2.2. Ivankar

Nuestro protagonista es un joven de 16 años, un mestizo de madre Zhule nacido en época de guerra, y ahora, sin hogar ni familia a la que regresar, se siente perdido en un mundo que lo maltrata y expulsa de la ciudad que lo vio crecer. Se siente maldito y desafortunado por todas las calumnias y desvergüenzas por las que se ha visto obligado a pasar.

Estadísticas:

Estadística	Valor base inicial	Crecimiento
Vida	150	A
Maná	50	B
Fuerza	10	A
Defensa	8	B
Intelecto	9	A
Defensa mágica	8	B
Velocidad	9	B

Tabla 22- Estadísticas base de Ivankar

5.1.3.2.3. Axel

Descripción: Amigo de Ivankar desde que se conocieron en el templo Zhule. De naturaleza valiente y decidida con unos grandes valores de su raza inculcados por los ancianos del templo. Sigue a Ivankar a todos lados pues está encargado de protegerle.

Estadísticas:

Estadística	Valor Base	Crecimiento
Vida	180	S
Maná	50	C
Fuerza	12	A
Defensa	10	B
Intelecto	6	C
Defensa mágica	10	B
Velocidad	7	B

Tabla 23- Estadísticas base de Axel

5.1.3.2.4. Leana

Descripción: Habitante de un poblado humano cercano al templo Zhule, estudiante de magia que quiere ver mundo y poner a prueba sus habilidades. De naturaleza amable, siente mucha simpatía por Ivankar.

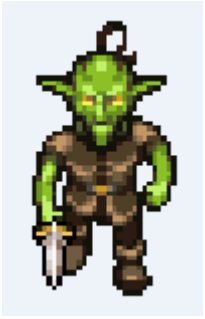
Estadísticas:

Estadística	Valor Base	Crecimiento
Vida	110	B
Maná	80	S
Fuerza	4	C
Defensa	5	C
Intelecto	10	S
Defensa mágica	10	A
Velocidad	7	B

Tabla 24- Estadísticas base de Leana

5.1.3.3. Enemigos

Los enemigos son los monstruos que nos aparecen a lo largo del juego, y éstos pueden aparecernos en cualquier lugar del mapamundi exterior y dentro de las mazmorras. Como los personajes, los enemigos también tienen sus propias estadísticas pero en contraparte a estos últimos, no crecen. A su vez, los enemigos tienen un añadido, que es la experiencia (tanto a nivel de personaje como de trabajo) y el oro que dan tras el transcurso de la batalla.

	Nombre: Goblin. Descripción: Pequeño humanoide de piel verde que suele acechar en zonas arboladas.	Vida: 120 Fuerza: 10 Defensa: 9 Intelecto: 5 Defensa mágica: 5 Velocidad: 8 Puntos exp: 5 Puntos trabajo: 4 Botín: 20 oro
	Nombre: Araña negra Descripción: Insecto de gran tamaño suele alternar entre bosques y cuevas.	Vida: 100 Fuerza: 6 Defensa: 5 Intelecto: 7 Defensa mágica: 10 Velocidad: 7 Puntos exp: 4 Puntos trabajo: 3 Botín: 15 oro
	Nombre: Murciélago Descripción: Habita en cuevas y se alimenta principalmente de sangre.	Vida: 100 Fuerza: 9 Defensa: 8 Intelecto: 2 Defensa mágica: 8 Velocidad:7 Puntos exp: 4 Puntos trabajo: 2 Botín: 5 oro

	Nombre: Esqueleto fantasma Descripción: Esqueleto donde reside el alma de antiguos guerreros.	Vida: 100 Fuerza: 10 Defensa: 7 Intelecto: 2 Defensa mágica: 13 Velocidad:12 Puntos exp: 6 Puntos trabajo: 4 Botín: 5 oro
	Nombre: Escarabajo gris Descripción: De los más débiles de su especie, consigue sobrevivir escondiéndose del resto de depredadores aunque suele ser agresivo con los humanos.	Vida: 30 Fuerza: 5 Defensa: 5 Intelecto: 5 Defensa mágica: 5 Velocidad: 5 Puntos exp: 3 Puntos trabajo: 3 Botín: 10 oro
	Nombre: Caballero Hok Descripción: Asesinado por sus compañeros, el odio de Hok hizo que su alma perdurara en el mundo, aunque en profundo sueño.	Vida: 500 Fuerza: 18 Defensa: 10 Intelecto: 0 Defensa mágica: 6 Velocidad: 8 Puntos exp: 100 Puntos trabajo: 20 Botín: 100 oro

Tabla 25- Enemigos disponibles

5.1.4 Niveles

En este apartado se incluyen los niveles que hay pensados para el juego.

5.1.4.1 Mapamundi

El mapamundi conecta las distintas ciudades y mazmorras.
En este caso conecta el pueblo Heufer (al oeste) con las minas abandonas (oeste).

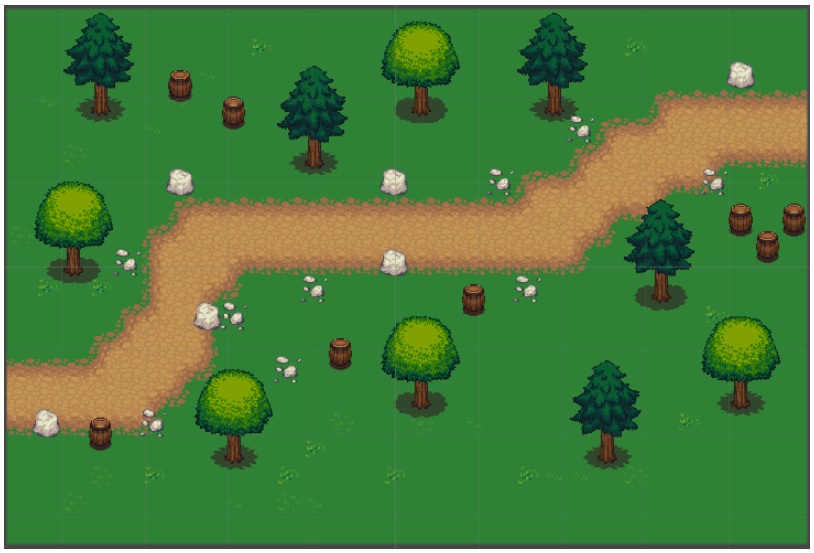


Figura 12- Muestra de mapamundi

5.1.4.2 Pueblo Heufer

Este pueblo de reciente construcción se encuentra en los límites del Novum Imperium. De momento tiene pocos aldeanos pero son todos de naturaleza amable. Aquí Ivankar escuchará información que le hará ayudar a estos aldeanos que tienen su principal recurso en las minas que hay al este la ciudad.

Figura 13- Pueblo Heufer

Figura 14- Interiores de Pueblo Heufer

5.1.4.3 Mina olvidada

Situada al este de pueblo Heufer, es la principal fuente de ingresos de este reciente pueblo, que descubrieron una nueva fuente de minerales pese a que la mina como ellos la llaman estaba abandonada.

Aunque al principio de explorarla les fue bastante bien a los pocos mineros del pueblo, poco a poco fueron teniendo problemas, pues había monstruos que les atacaba y es que realmente el lugar no es una mina. Es donde descansa un antiguo guerrero que pereció hace tiempo, pero su odio a sus asesinos hicieron que su alma perdurara en el mundo, dentro de su antigua armadura.

El jugador debe de avanzar por la mazmorra hasta llegar al final donde se encuentra el jefe y derrotarlo.

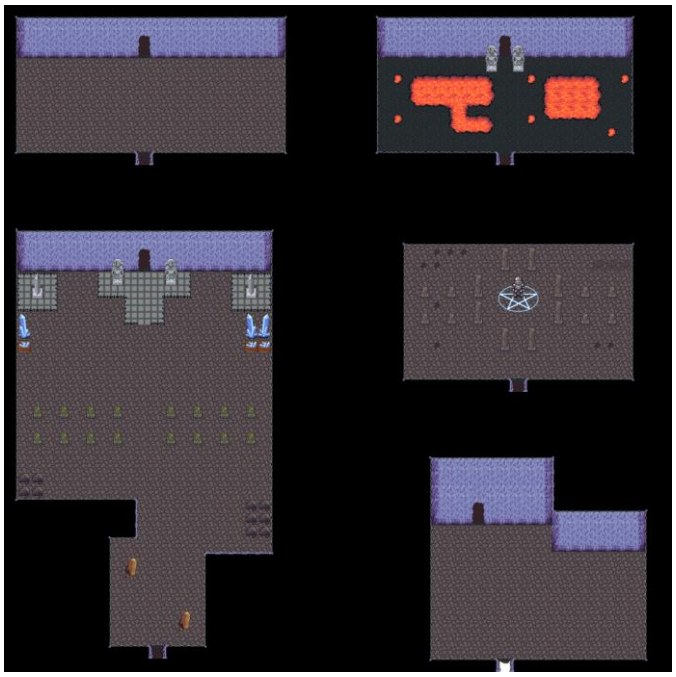


Figura 15- Secciones Mina olvidada

5.1.5. Interfaz

5.1.5.1. Sistema visual

En este apartado se muestran los diversos elementos visuales de los que dispone el juego como tipos de gráficos, menús, etc.

5.1.5.1.1. Menús

Hay una gran cantidad de menús, desde el principal con las opciones de "Nueva Partida" y "Continuar" a los menús dentro del juego.
Una vez empezada la partida, se encuentra el menú principal de pausa, donde se encuentran las opciones de "inventario" (se ven los objetos actualmente en posesión del jugador), "trabajos" (se ven los trabajos obtenidos hasta ahora), "grupo" (se puede ver a los integrantes del grupo con sus estadísticas).

5.1.5.1.2. Estilo visual

Los gráficos del juego están compuestos de sprites.
Dependiendo del evento, la forma de los éstos cambia, como por ejemplo a la hora de moverse por el mapa donde el del personaje hace una animación de movimiento.
En combate se pueden ver los sprites de los enemigos así como los retratos de los personajes del grupo.

5.1.5.1.3. Cámara

Hay dos tipos de cámara: cenital y frontal. La cenital se usa para seguir al personaje a lo largo del mapa, ya sean ciudades, mazmorras o mapamundi. La frontal se utiliza para los combates donde al fondo/arriba están los enemigos y al frente/abajo están los personajes junto con la barra de tiempo y sus estadísticas de vida y maná visibles.

5.1.5.2. Sonido

Para dotar al juego de un mayor grado de profundidad es importante aportar una buena OST. Para el menú de inicio sonará el tema principal del juego. Durante las animaciones, el tema que suena se corresponderá a los eventos principales de las historia. Por ejemplo, si estamos en una situación de tensión será más tensa, mientras que en una situación triste la música será más pausada y emotiva. Para los combates habrá un tema específico. Luego para habrá una recopilación de sonidos para las diferentes acciones (animación combate, navegar por los menús, mover el cursor).

5.1.6. Inteligencia artificial

5.1.6.1. Inteligencia artificial de los enemigos

En la batalla, la inteligencia artificial de los enemigos consistirá en buscar al miembro más débil y atacarle hasta que muere.

5.1.6.2. Inteligencia artificial NPC

Los NPC's del juego solo aportan diálogo por lo que sus comportamiento dentro del juego aportar información. Su movimiento en las cinemáticas está predefinido y no lo pueden alterar.

5.2. Requisitos funcionales y no funcionales

A continuación, como información complementaria al documento de diseño del juego, se exponen los requisitos funcionales y no funcionales para el juego.

5.2.1 Requisitos funcionales

- Movimiento del personaje: El jugador tiene que poder mover al personaje por el mapa y de esta forma recorrer las distintas mazmorras y ciudades.
- Sistema de trabajos: El jugador debe poder equipar o asociar un trabajo/clase a los personajes. Estos trabajos deben proporcionar una modificación de las estadísticas y unas habilidades determinadas.
- Batalla ATB: La batalla debe de estar compuesta por barras de tiempo (una por unidad en batalla), donde al llenarse cada barra se detendrá momentáneamente el flujo de la batalla para decidir qué acción hace cada unidad.
- Transición de niveles: Al cambiar de mapa (ciudad-mapamundi, mapamundi-ciudad…) se produce una transición donde se produce un efecto de fade-in / fade-out donde la pantalla se oscurece primeramente y se hace más clara al cargar el siguiente mapa.
- Uso de objetos/habilidades: El jugador debe de poder utilizar objetos tales como pociones (para restaurar puntos de vida) y habilidades (proporcionadas por los trabajos).
- Conversaciones: Se debe poder interactuar con los NPC y poder leer las conversaciones que sucedan con éstos.
- Guardar/Cargar: El sistema debe poder guardar y cargar los datos de la partida para continuarla en cualquier momento.

5.2.2. Requisitos no funcionales

- Lenguaje C#: La programación del juego se llevará a cabo en el lenguaje de programación C#.
- Idioma español: Los textos del juego estarán en castellano.
- Controles de mando: Los controles del juego se adaptarán para que se pueda jugar con mando.

5.3. Estructura del juego

Como bien se ha podido ver en la figura X, el juego hace soporte a distintas funcionalidades, y en este apartado se va a explicar en qué consiste cada apartado.

5.3.1. GameManager

El GameManager es una clase siempre presente a lo largo de la ejecución del juego, y es la encargada de controlar el ciclo de ejecución del juego, así como el estado de éste (pausa, batalla, mapa…) ya que puede cambiarlo cuando es necesario. Además, está asociado con el resto de clases que almacenan datos como las bibliotecas y el grupo de personajes. También tiene control sobre parte de la interfaz del juego (menús) ya que es capaz de activarlos o desactivarlos.

En resumen es la clase encargada de controlar todo lo que pasa en el juego, así como de inicializar todos los componentes que permiten que funcione el juego.

5.3.2. Controlador de batalla

El controlador de batalla es la clase que se encarga de inicializar todos los elementos que intervienen en la batalla (unidades, interfaz…) y de controlar el flujo de éste con una secuencia de estados (espera, decisiones de jugador/enemigo, selección de objetivo, cálculo de daño, victoria, derrota y escapar).

El flujo de la batalla es el siguiente: primeramente la batalla está en modo espera, donde se llenan las barras de tiempo y en el momento se llena una de las barras, pasa al siguiente estado (elección de jugador o elección de enemigo) donde se elige la acción a realizar. Seguidamente, se selecciona el objetivo y una vez hecho esto, se pasa al cálculo de daño para poder aplicarlo. Una vez aplicado el daño al/los objetivos pertinentes, se comprueba la condición de victoria/derrota y en caso de cumplirse se pasa a ese estado que da fin a la batalla.

5.3.3. Personajes jugables y crecimiento de estadísticas

Los personajes jugables son los personajes que el jugador puede controlar, es decir, pertenecer al grupo. Estos se caracterizan por poder tener, además de las estadísticas base, un trabajo asociado que potencia sus estadísticas. Así pues, también pueden subir de nivel, lo cual hace que sus estadísticas crezcan. El crecimiento de estadísticas se aplica mediante probabilidades, es decir, dependiendo de la estadística y su grado de crecimiento, tendrá una probabilidad de sumar más o menos puntos.

5.3.4. Enemigos

Los enemigos están definidos, al igual que los personajes por una serie de estadísticas base, diferenciados unos de otros por su nombre y además por la zona en la que aparecen. Ya que dependiendo del valor de ésta podrán aparecer distintos enemigos a la hora de la batalla

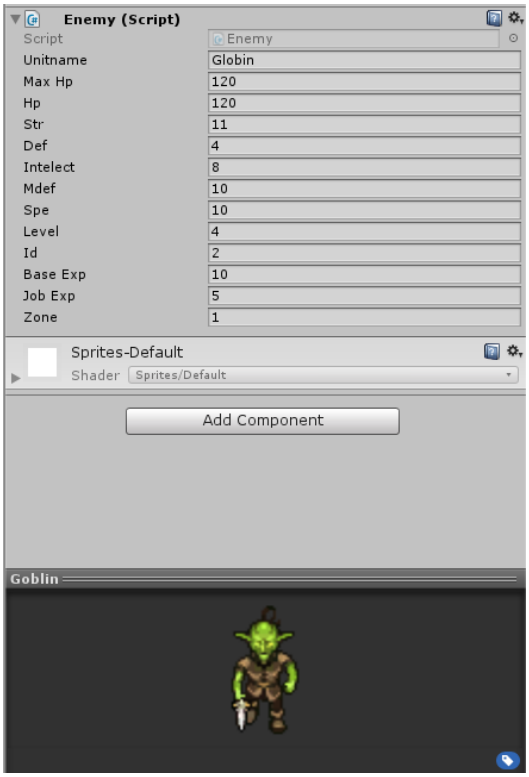


Figura 16- Ejemplo de enemigo en el inspector de Unity

5.3.5. Jugador y cámara

El jugador como tal es únicamente el sprite que se mueve por pantalla, pero es importante puesto que siempre está presente a lo largo de la ejecución.
La cámara está relacionada con el jugador, puesto que le sigue en todo momento.

5.3.6. Trabajos y habilidades

Trabajos y habilidades van de la mano, ya que los trabajos proporcionan habilidades a los personajes y las habilidades están asociadas a un único trabajo.

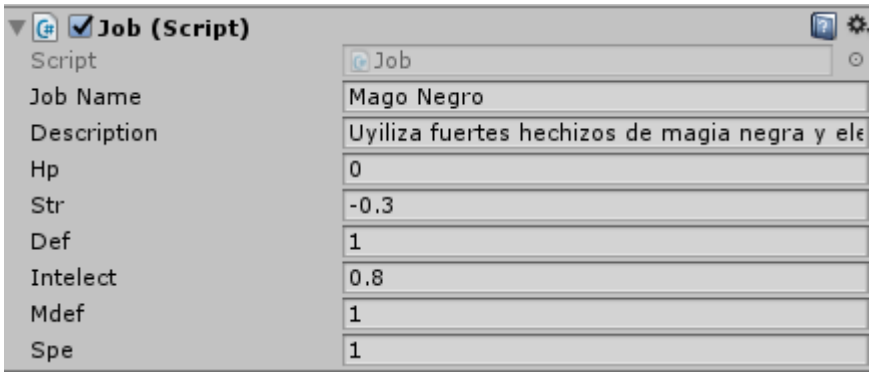


Figura 17- Ejemplo de trabajo el inspector de Unity

Como puede observarse en la figura 16, la función principal del trabajo es la de realizar una modificación de las estadísticas de los personajes (hp, str, def, intelect, mdef y spe son los modificadores).

Por lo que respecta a las habilidades, como se puede observar en la figura 17, están definidas por la potencia de esta, mana que cuesta, trabajo al que pertenecen y el tipo (daño, curación…) entre otras cosas.

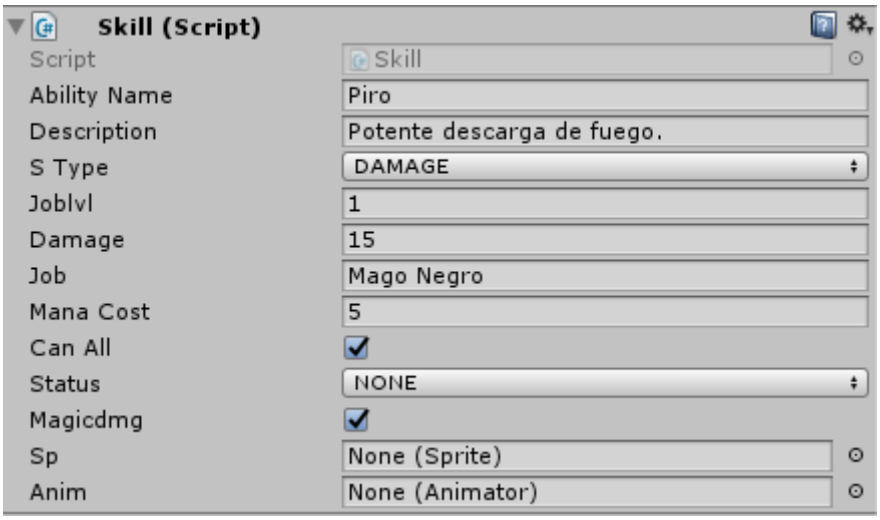


Figura 18- Ejemplo de habilidad en el inspector de Unity

5.3.7 Bibliotecas

Las bibliotecas almacenan los datos de diferentes colecciones como los enemigos, los trabajos, las habilidades, los objetivos y los objetos.

Se cargan una vez al principio del juego y son siempre accesibles. Dentro de las bibliotecas se pueden diferenciar dos tipos: las que se cargan como desde fichero y las que no.

Las que se cargan desde fichero son las bibliotecas de objetos y la de objetivos. En este caso se trata de un fichero json, que contiene los datos que identifican a cada objeto y a cada objetivo.

El resto son las que no se cargan desde fichero, se cargan como una instancia de prefab (ver apartado 5.4).

Pasando ya al funcionamiento de las bibliotecas, todos los datos se guardan en una lista contenida en el script que actúa como biblioteca en cada caso. Funcionan como bases de datos, es decir, en base a una consulta realizada mediante un método en concreto (buscar por ID, por nombre, por tipo…) devolverá el o los resultados que se correspondan con el criterio de búsqueda utilizado.

5.3.7.1 Objetivos

Los objetivos marcan la línea de acción del juego, es decir, a donde debe ir el jugador, si el jugador debe de obtener algún objeto…
Así pues, los objetivos están pensados principalmente para funcionar en una estructura similar a la de un árbol, es decir, un objetivo puede desbloquear uno o más objetivos y a su vez un objetivo puede necesitar de uno o más objetivos completados para ser desbloqueado.

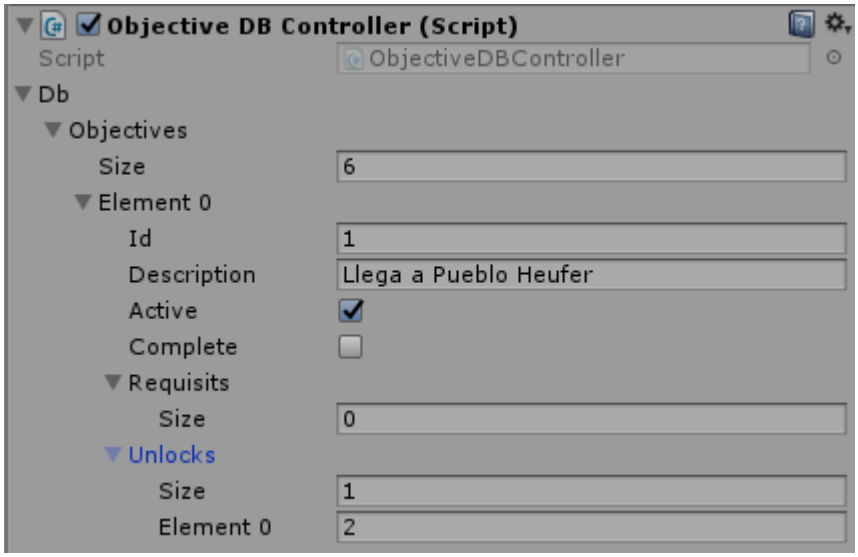


Figura 19- Ejemplo de Objetivo en el inspector de Unity

5.3.7.2. Objetos

Los objetos en el juego ayudan al jugador a avanzar por el contenido. Están definidos de forma genérica para que a partir de un solo script se puedan generar distintos tipos de objetos.

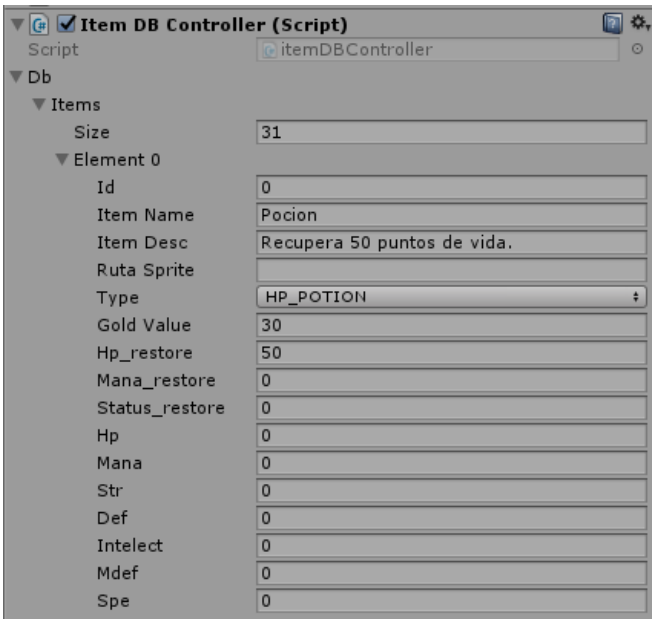


Figura 20- Ejemplo de objeto en el inspector de Unity

5.4. Prefabs

En este apartado se habla de los prefabs que han sido necesarios para la elaboración del videojuego.

5.4.1. Prefabs que actúan como bases de datos

Estos prefabs como se ha mencionado anteriormente, forman parte de las bibliotecas del juego. Están compuestos por un gameObject raíz que contiene el script (manager) que hace la función de biblioteca y diversos gameObject por debajo del elemento raíz, es decir, los hijos que contienen a su vez el script del tipo de dato pertinente. El funcionamiento es el mismo en los 3 casos, primeramente, al instanciarse el prefab, el script (manager) de cada uno obtiene los datos de los scripts de los hijos y los guarda en una lista.

5.4.1.1. Enemigos

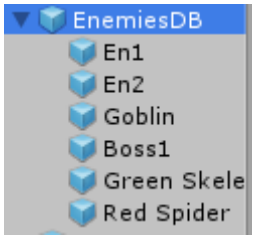


Figura 21- Vista prebaf Enemigos

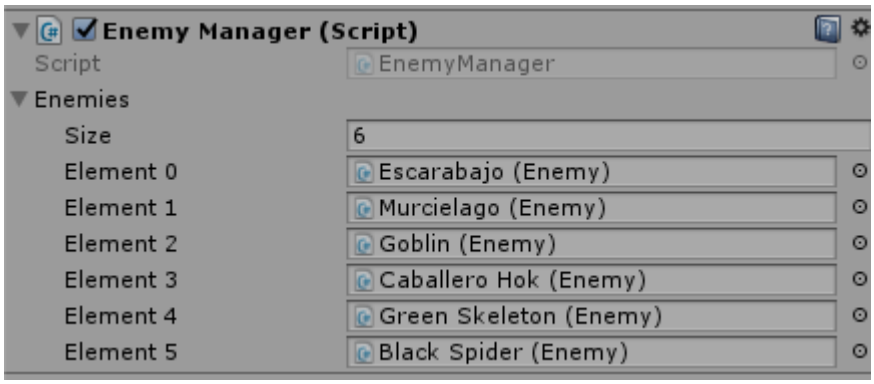


Figura 22- Vista Script EnemyManager en el inspector de Unity

5.4.1.2. Trabajos

Figura 23- Vista del prefab de trabajos

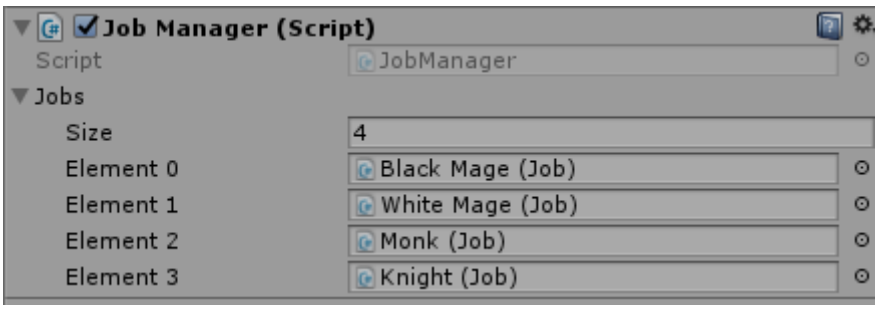


Figura 24- Vista script JobManager en el inspector de Unity

5.4.1.3. Habilidades

Figura 25- Vista parcial del prefab de habilidades

Figura 26- Vista del script SkillManager en el inspector de Unity

5.4.2 Trasladores

A lo largo del juego, el jugador cambia de localización (de mapamundi a ciudad o a mazmorra por ejemplo), así pues los puntos en los que se cambia de localización (y de escena) están guardados como prefab. La necesidad de estos prefab estaba en que no era posible poner las coordenadas (x, y) del destino a mano, ya que debido a la organización de los gameObject y su posición de referencia respecto a la escena podía no ser la misma.

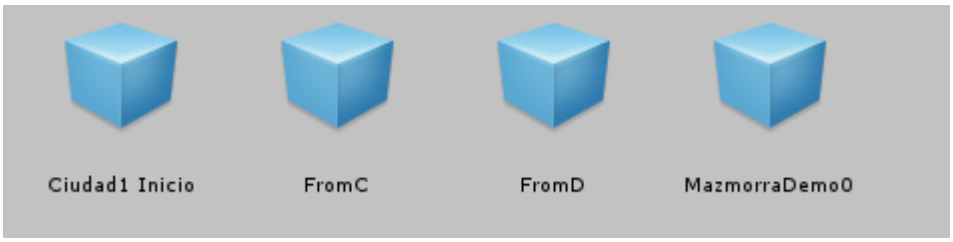


Figura 27- Prefabs trasladores

5.4.3. Prefabs de interfaz

Los prefabs de interfaz son muy simples pues suelen contener pocos elementos de tipo UI (textos, paneles…). Su función es principalmente ayudar a mostrar la información por pantalla. Esto se consigue instanciando estos prefabs mediante código rellenando así su panel contenedor.

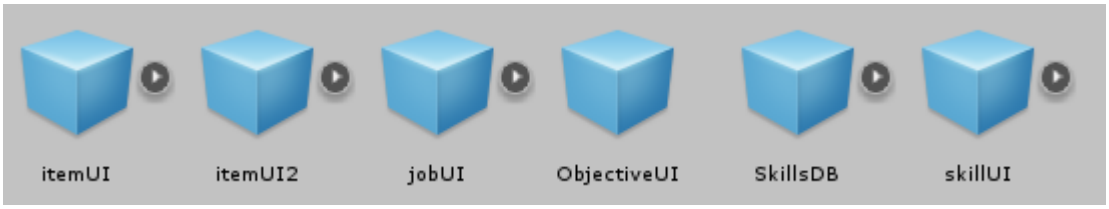


Figura 28- Prefabs de interfaz

5.4.4. Prefab de grupo

El prefab de grupo almacena todos los personajes jugables del juego.

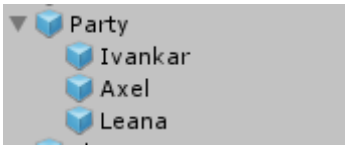


Figura 29- Prefab de grupo

5.5. Serialización de datos

La serialización de datos se utiliza para poder guardar datos del juego dentro de un fichero (en nuestro caso, un fichero json).

El guardado de datos mediante serialización es muy fácil de hacer puesto que solo debemos indicar que la clase que queremos guardar en fichero sea serializable.
Aunque no permite serializar tipos de datos complejos como pueden ser los diccionarios, permite guardar todo tipo de datos comunes como int, float, string, listas… y en caso de necesitar que se guarde información de datos que no admiten serialización se puede hacer un script para descomponer esa estructura más compleja, extraer sus datos simples y poder guardarlos. Y a la hora de carga en este caso específico es hacer el proceso inverso cargar los datos del fichero en la estructura.

El proceso de serialización de datos se compone de una serie de llamadas a las funciones de la api de json que vienen por defecto en Unity y de forma automática, indicándole el destino (fichero en caso de hacer guardado y variable en caso de hacer cargado de datos), se hacen todos los procesos para ambas operaciones.

5.6. Creación de niveles

Al ser un videojuego en 2D, la herramienta utilizada para elaborar los distintos niveles/mapas que componen el juego ha sido Tiled.

Esta herramienta permite dividir el escenario en capas pudiendo así ofrecer efectos de profundidad además de tener así organizados los distintos elementos del mapa.

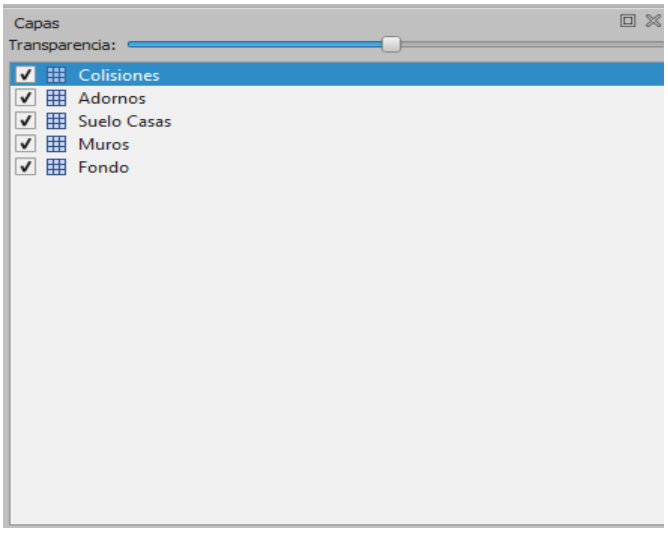


Figura 30- Capas de edición en Tiled

A su vez, también podemos establecer las colisiones existentes con Tiled, donde posteriormente al exportar se transformarán en *colliders* de Unity.

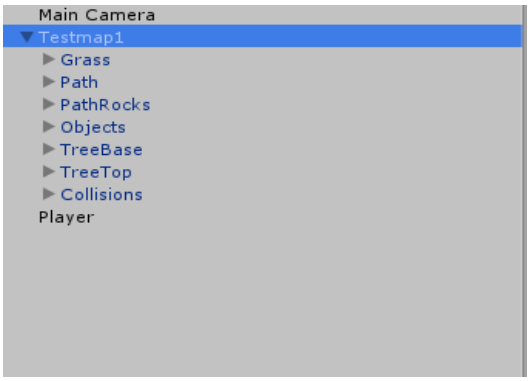


Figura 31- Resultado de exportación de mapa en Unity

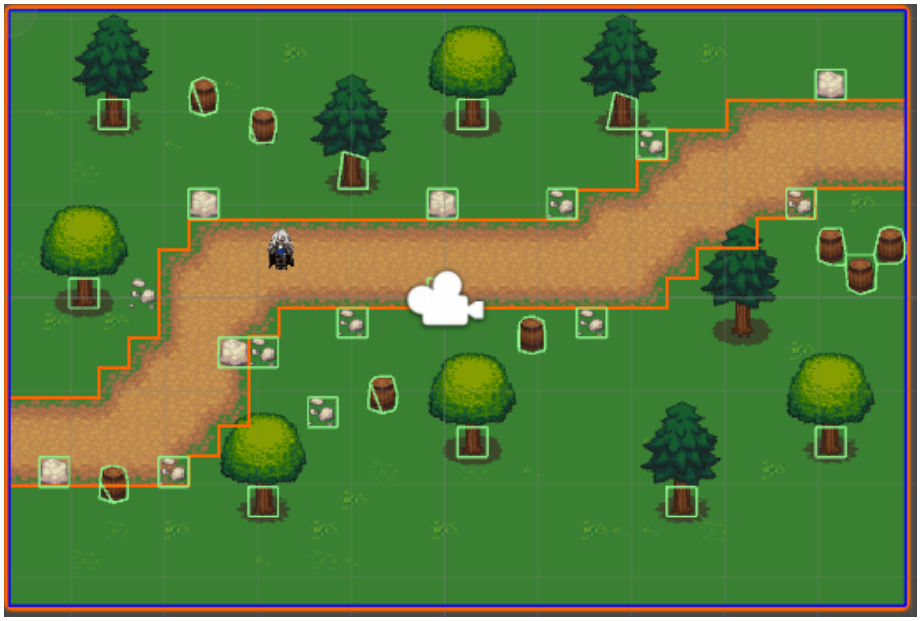


Figura 32- Vista de las colisiones en el mapa exportado

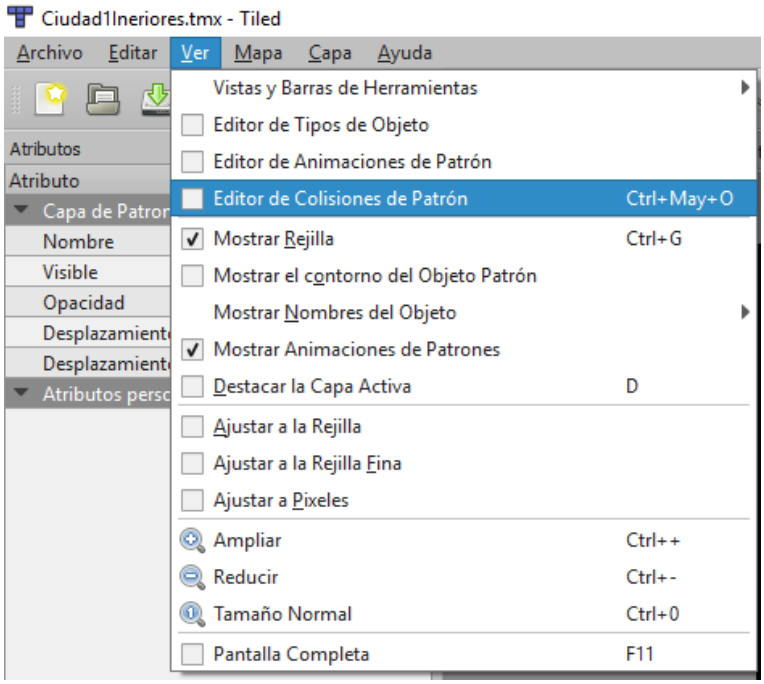


Figura 33- Vista del menú de Tiled para crear colisiones

Para crear la colisión en Tiled, entramos en el Editor de Colisiones de Patrón y seguidamente se selecciona uno de los tiles que tengamos cargados, en este caso se ha utilizado un sprite de 32x32 de color rojo en su totalidad. Para terminar se selecciona la forma que queremos que tenga la colisión (personalizada, cuadrado, círculo) y seleccionamos todo el sprite con la forma seleccionada.

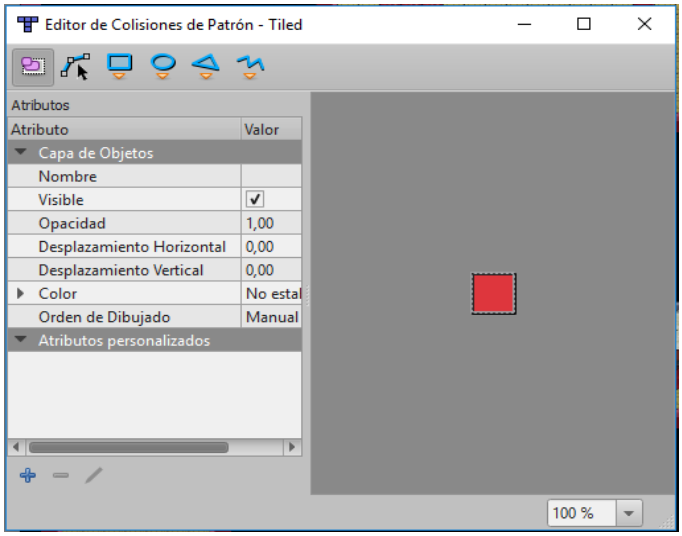


Figura 34- Editor de Colisiones de Tiled

Figura 35- Colisiones en Tiled (1)

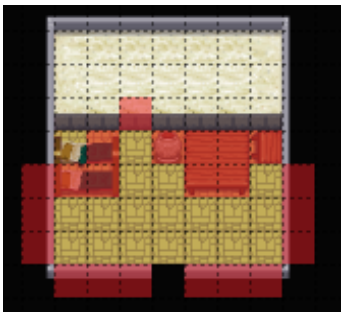


Figura 36- Colisiones en Tiled (2)

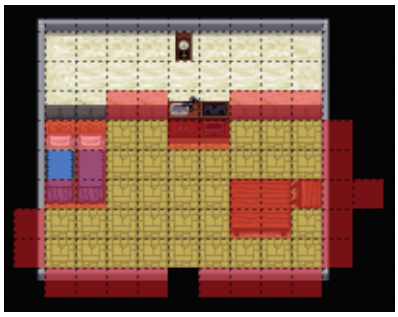


Figura 37- Colisiones en Tiled(3)

Para exportar los mapas creados en Tiled, se ha utilizado una herramienta gratuita llamada Tiled2Unity, que permite en un solo "click" convertir el archivo *.tmx* en un *prefab* de Unity.

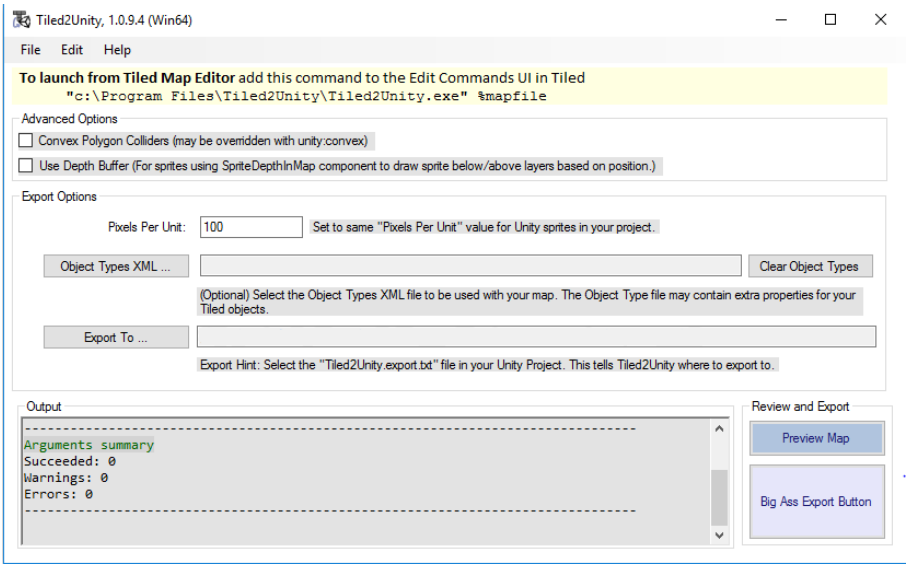


Figura 38- Interfaz de Tiled2Unity

El detalle más importante a tener en cuenta es el parámetro *Pixels Per Unit*, el cual hay que poner a 100 para que tenga la escala correcta y pueda visualizarse el mapa como realmente es.

5.7. Animaciones

Las animaciones se han hecho mediante el animador de Unity. El proceso consiste en primeramente cortar la imagen, es decir, dividir la imagen donde se encuentran todos los movimientos de un personaje en imágenes más pequeñas que contienen un único paso (p.ej: pie derecho adelante mirando a la izquierda).

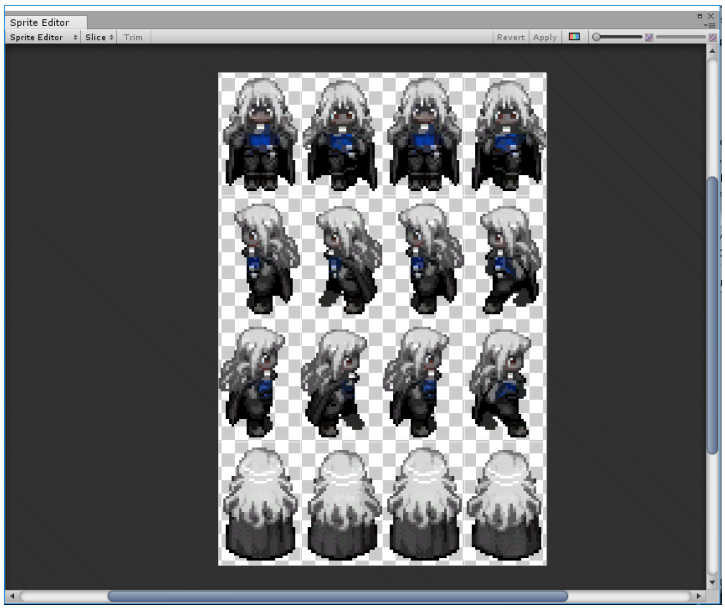


Figura 39- Editor de sprites de Unity

Seguidamente, se crea un “animator controller”, que nos proporciona una máquina de estados interna mediante la cual se puede cambiar la animación del personaje. Esto es posible gracias a la adición de estados y variables para realizar la transición de un estado a otro.

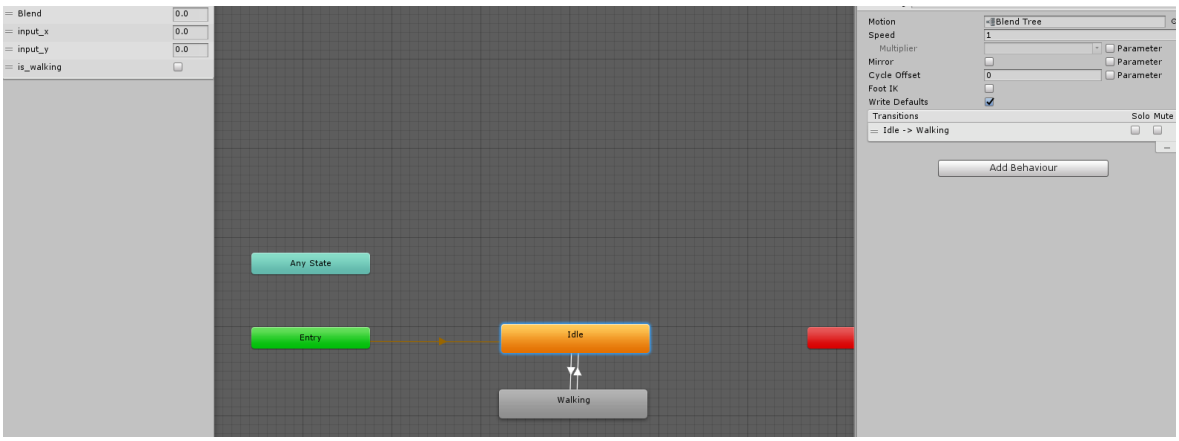


Figura 40- Variables y estados del Animador

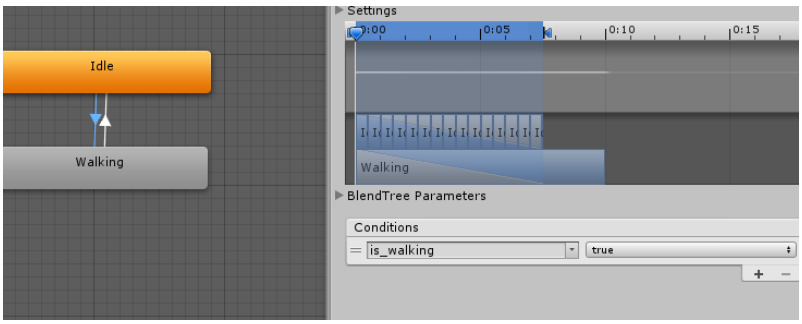


Figura 41- Ejemplo de transición entre estados

Y dentro de cada estado se utilizan las variables para cambiar el sprite de nuestro personaje. Como se puede observar en la figura 36, en función del input nos puede llevar a 4 distintas situaciones, donde en cada una de ellas se muestra una sprite (en este caso el personaje detenido mirando en alguna de las 4 direcciones).

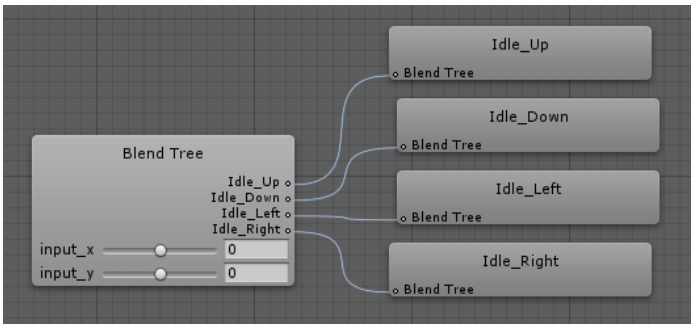


Figura 42- Funcionamiento interno de un estado

Y de esta forma se establece la forma en la que se establece el esqueleto de las animaciones. A continuación, para crear la animación propiamente dicha, hay que seleccionar primeramente los sprites que van a formar parte de ella y posteriormente crear una animación. Una vez este creada, hay que ir al editor de animaciones, para o bien comprobar que esta todo correcto o bien para hacer la animación más larga (en vez de una animación de 4 frames, hacer una de 8) o bien para poner los sprites en el orden que se quiera.

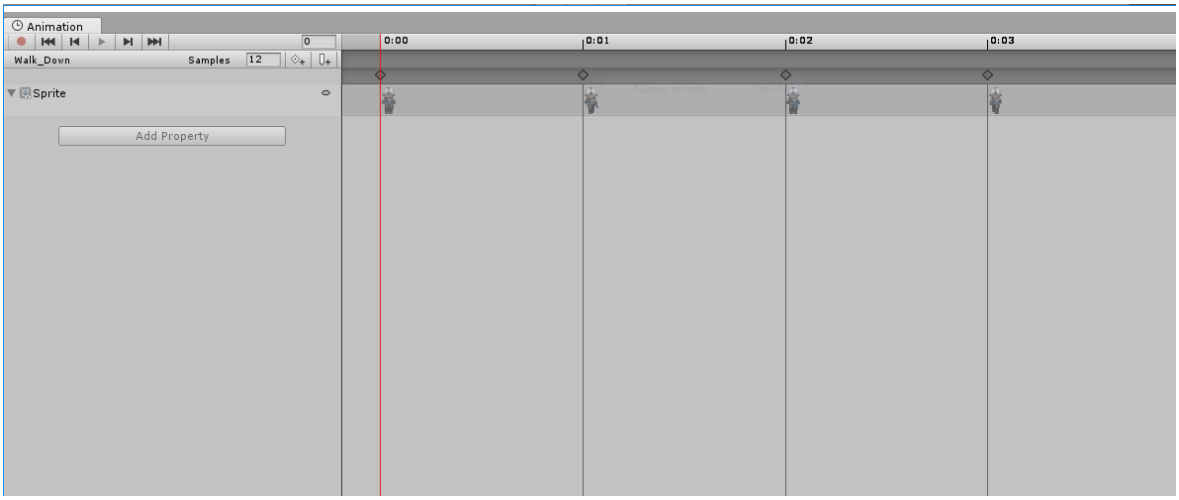


Figura 43- Editor de animaciones de Unity

Posteriormente, mediante código se establece la lógica que llamará a los estados de la máquina del "animator controller" para que al mismo tiempo que se actualiza la posición del personaje se produzca la animación mientras se introduzca el input necesario.

5.8. Asset de diálogos RPGTalk

RPGTalk es un asset de la tienda de Unity desarrollado por Seizestudios, completamente gratuito, que ofrece al desarrollador una forma muy fácil de crear diálogos con varios efectos como la animación de texto, la animación de los retratos al producirse el retrato,...

Para utilizar este asset lo único que hay que hacer es arrastrar el prefab que nos trae el asset y preparar la escena con un canvas y los diferentes paneles que componen la UI del diálogo.

Figura 44- Estructura de elementos UI para RPGTalk

A continuación se pueden ver los distintos parámetros/variables del asset junto con la multitud de opciones disponibles.

El primer parámetro es el más importante puesto que sin él, el diálogo no podría tener lugar. Estamos hablando del fichero "*.txt*" que contiene el texto del diálogo.

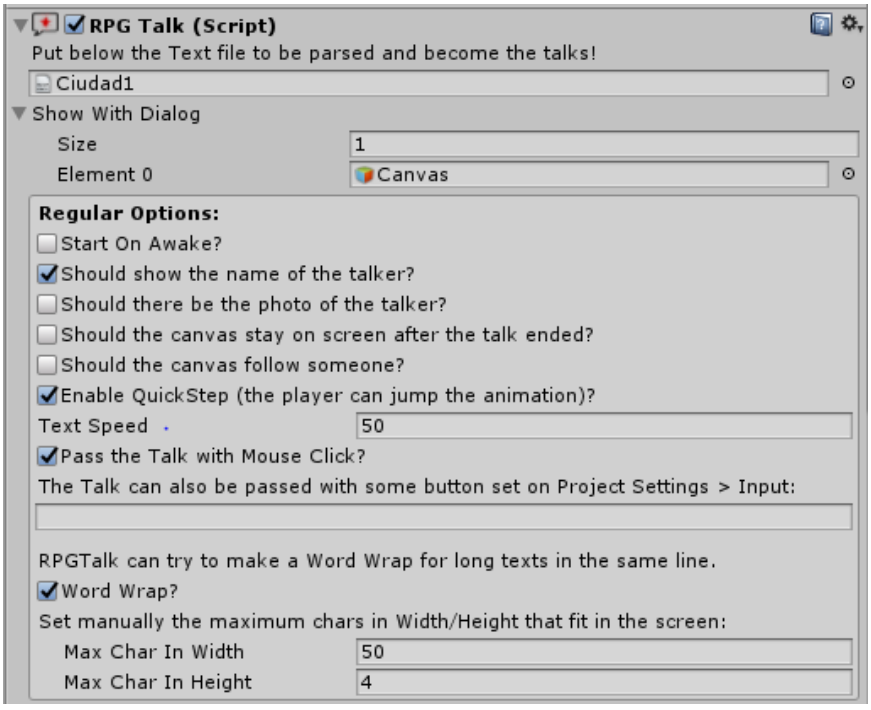


Figura 45- RPGTalk en inspector de Unity (1)

Seguido del fichero, se pueden ver distintas opciones y parámetros como el/los elemento/s UI al que debe estar enlazado el texto.
Por lo que respecta a las opciones, permiten bastante personalización como: empezar el diálogo al empezar la escena, mostrar o no el nombre del personaje que está hablando en ese momento, mostrar

o no el retrato del personaje mientras habla, mantener el canvas una vez ha terminado la conversación, hacer que el canvas siga o no a un personaje para que el texto esté junto al personaje en todo momento, la velocidad del texto, el input con el que el jugador puede continuar con el diálogo una vez se ha llenado el cuadro de texto y hacer que el texto se ajuste al tamaño del cuadro texto.

Siguen los bloques de interfaz, animación al hablar, audio y el bloque de callbacks, scripts y variables, que automatiza la llamada a otro método en caso de necesitarlo y con las variables se logra más personalización. Por ejemplo: si el jugador puede elegir su nombre, mediante una variable nombrada por ejemplo "nombreJugador" se puede mostrar el nombre deseado en el cuadro de diálogo. Además se pueden utilizar estas variables dentro del fichero de texto para de esa forma hacer los cambios en nombres o en cualquier parte del texto.

Figura 46- RPGTalk en inspector de Unity (2)

Para la interfaz se deben de enlazar los elementos UI en los cuales va a estar contenido el nombre del personaje, el texto del diálogo, los retratos de los personajes (en caso de estar activada la opción) y un objeto que indique la continuación del diálogo como por ejemplo una flecha.
En el bloque de callbacks, además de lo mencionado anteriormente, es donde se indica el principio y fin del diálogo, es decir, se indican las líneas del fichero que se deben leer.
Finalmente se pueden añadir efectos de audio al diálogo tanto al pasar cada letra como al avanzar en el diálogo cuando el jugador pulsa el input correspondiente.

Figura 47- Ejemplo de cuadro de diálogo en RPGTalk

5.9. Editor de inventario en el inspector de Unity

Para realizar pruebas en el inventario de una forma más rápida, se ha desarrollado una pequeña modificación del inspector de Unity para de esta forma poder hacer pruebas más visuales y además en tiempo de ejecución.

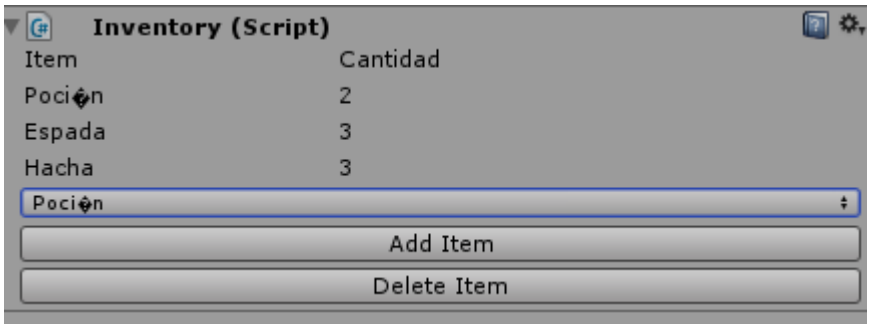


Figura 48- Editor de inventario en inspector de Unity

Se puede ver como se listan los elementos del inventario y además un desplegable en el que se muestran las posibles opciones de los objetos que pueden estar en el inventario. Dichos objetos que están en el desplegable son recogidos de la base de datos creada a partir de un fichero json. Y por último los botones que permiten añadir o eliminar del inventario el objeto seleccionado mediante la lista desplegable.

5.10. Elaboración de los menús

En este apartado se exponen los distintos menús del juego y como han sido creados.

5.10.1. Menú principal

El menú principal, o el menú de pausa es el que contiene el acceso a todos los submenús (trabajos, grupos, objetivos, inventario…). El menú consiste en un panel contenedor y una cantidad de 7 botones:

- Continuar: Se puede continuar con la aventura al quitar al modo pausa.
- Inventario: Se activa el panel de interfaz de inventario que muestra los objetos que hay en el inventario.
- Grupo: Se activa el panel que contiene la información del grupo.
- Trabajos: Se activa el panel que contiene la lista de trabajos disponibles.
- Objetivos: Se activa el panel que contiene la lista de objetivos activos.
- Guardar: Guarda los datos de la partida.
- Salir: Salir del juego.

Figura 49- Menú principal

5.10.2. Menú de inventario.

Este menú muestra todos aquellos objetos que están en el inventario. El menú está compuesto por un panel contenedor de los elementos UI. Seguidamente descendiendo por el árbol de la escena, hay otro panel donde se le aplica el componente de scroll y por último, el panel donde se introduce el contenido que tiene asociado un componente "vertical layout group" para que los bloques de contenido se pongan uno debajo del otro.

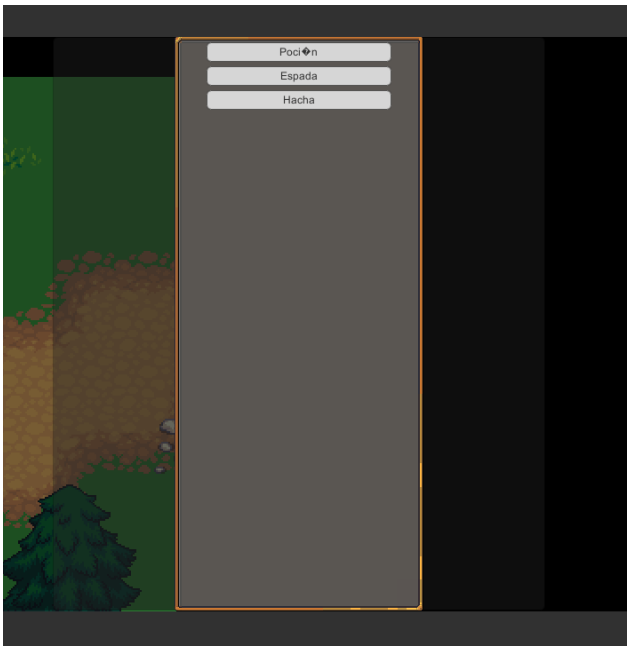


Figura 50- Menú inventario

Además, si se clica en cada objeto del inventario, aparece un menú a la derecha con el nombre de los personajes del grupo, donde se podrá o bien usar una poción con el seleccionado o bien equipar un objeto, todo en función del objeto seleccionado previamente.

5.10.3. Menú de grupo.

Este menú muestra el estado actual del grupo mostrando los 3 primeros miembros con sus estadísticas, trabajo, nivel y vida además de su nombre y retrato.

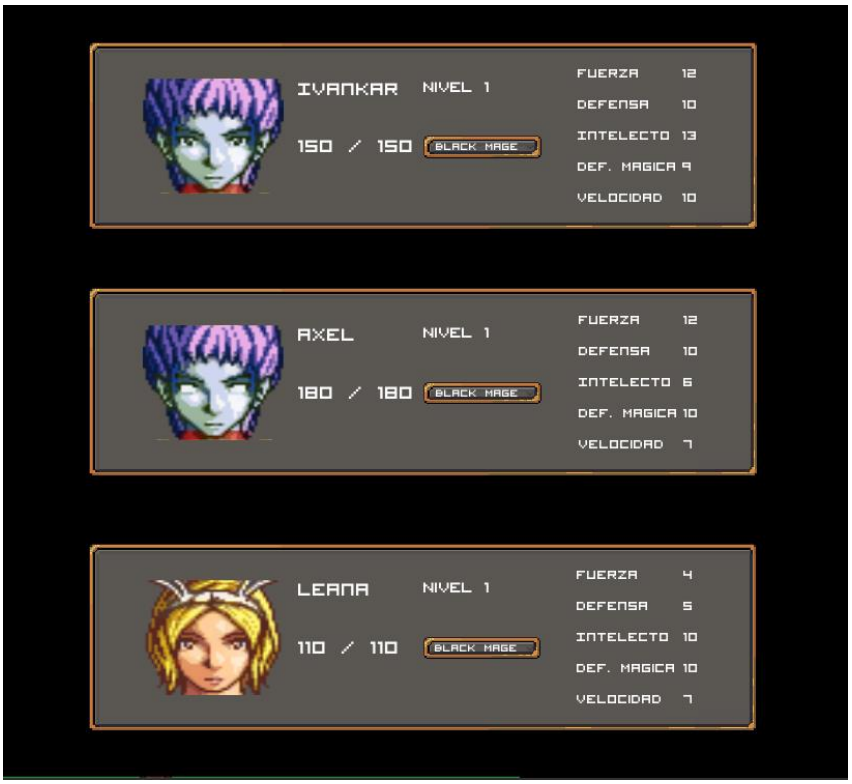


Figura 51- Menú de grupo

Está compuesto por un panel raíz donde se agrupan todos los componentes UI del menú. Seguidamente hay 3 paneles contendores, 1 por cada miembro a mostrar. Dentro de cada panel hay varios componentes de tipo texto (nombre, vida, nivel, estadísticas y valor de las estadísticas), un componente de imagen para los retratos y un menú desplegable donde se encuentran los trabajos disponibles. Y donde sí se pasa el ratón por alguna de las opciones se muestra un panel informando de las posibles nuevas estadísticas y en caso de seleccionar algún trabajo distinto al equipado, se produce el cambio de trabajo del personaje.

Figura 52- Ejemplo de vista previa de estadísticas

5.10.4. Menú de trabajos.

Este menú muestra la lista de trabajos disponibles y si clicamos encima de cada uno, en el panel de la derecha nos muestra más información sobre el trabajo seleccionado. El menú está compuesto por un panel contenedor de los elementos UI. Seguidamente descendiendo por el árbol de la escena, hay otro panel donde se le aplica el componente de scroll y por último, el panel donde se introduce el contenido que tiene asociado un componente “vertical layout group” para que los bloques de contenido se pongan uno debajo del otro. Además de otro panel adicional donde se muestra la información del trabajo seleccionado.

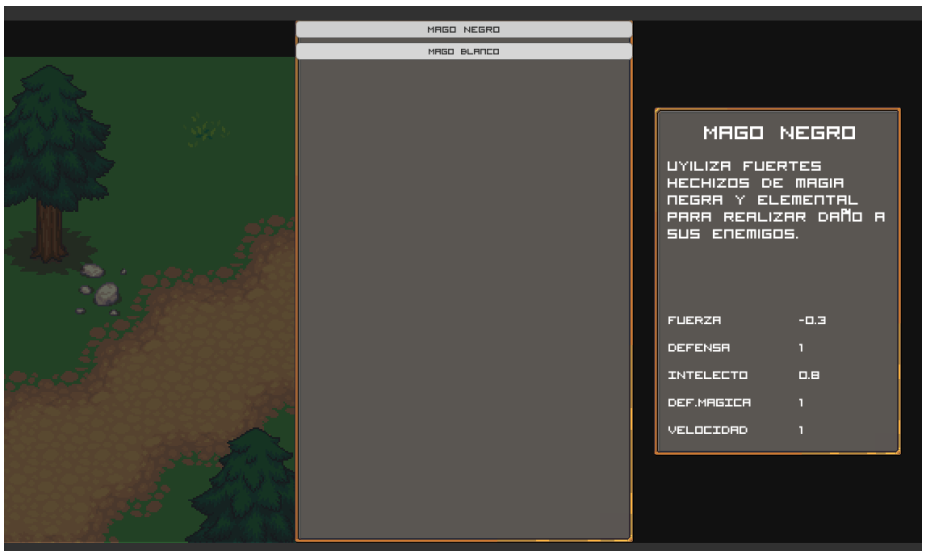


Figura 53- Menú trabajos

5.10.5. Menú de objetivos.

El menú de objetivos muestra los objetivos activos en ese momento. El menú está compuesto por un panel contenedor de los elementos UI. Seguidamente descendiendo por el árbol de la escena, hay otro panel donde se le aplica el componente de scroll y por último, el panel donde se introduce el contenido que tiene asociado un componente "vertical layout group" para que los bloques de contenido se pongan uno debajo del otro.

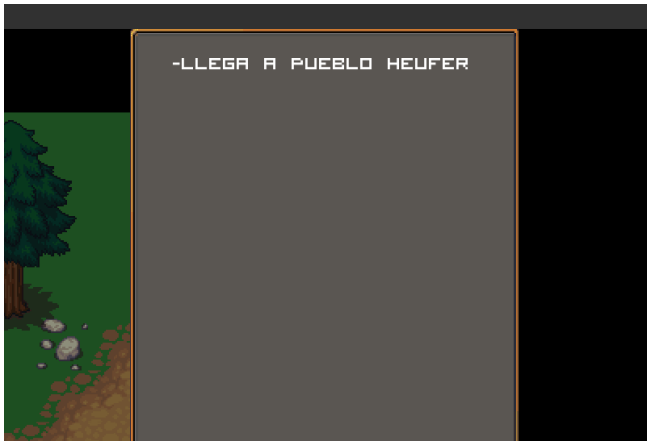


Figura 54- Menú objetivos.

5.11. Interfaz Batalla

La interfaz de batalla está diseñada para que se muestren el fondo, los retratos de los personajes, los sprites de los enemigos y las barras de tiempo de cada uno de ellos.

La interfaz consta de los siguientes elementos: una imagen para el fondo y 3 paneles contenedores (enemigos, personajes y acciones).

En el caso del panel de enemigos hay 3 paneles más que contienen los elementos de cada enemigo, donde hay dos imágenes, una para el sprite y otra para la barra de tiempo.

Respecto a los personajes, la organización es igual a la de los enemigos, con la diferencia de que hay textos para informar de la vida o maná.

Finalmente, el panel de acciones que contiene 4 botones que permiten realizar las 4 acciones de la batalla (atacar, habilidades, objetos y huir).

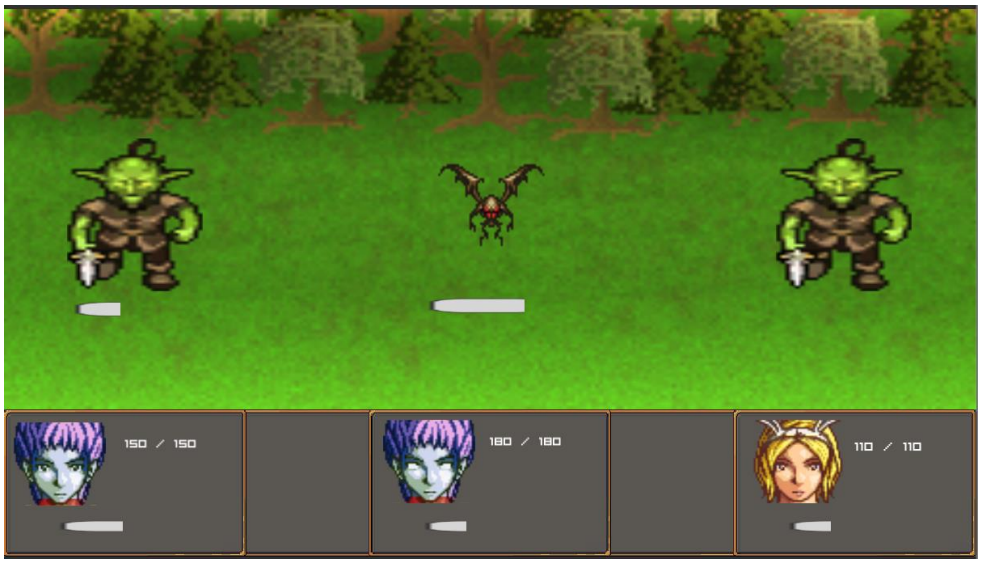


Figura 55- Interfaz de batalla

5.12. Arte del juego

En este apartado se recopilan todos los recursos artísticos utilizados para realizar este proyecto.

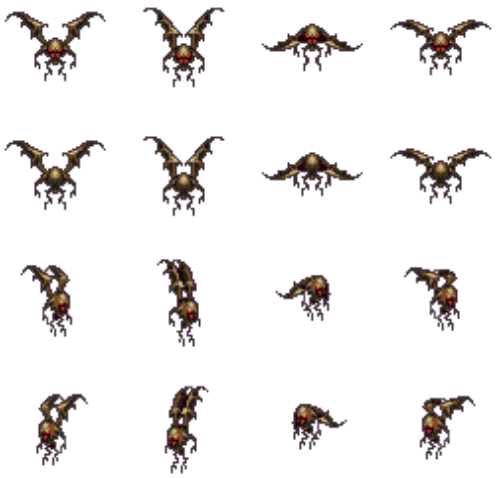


Figura 56- Spritesheet del enemigo Murciélago

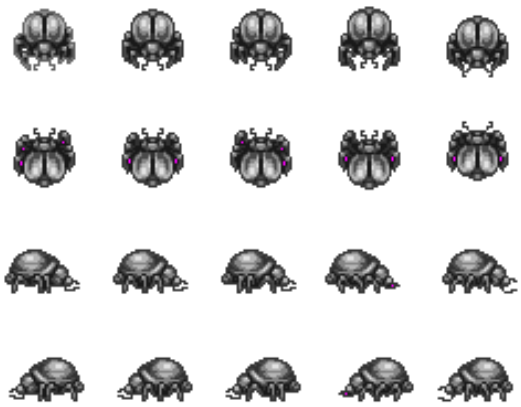


Figura 57- Spritesheet del enemigo Escarabajo Negro

Figura 58- Spritesheet de Aldeana tipo 1

Figura 59- Spritesheet de Axel

Figura 60- Spritesheet de Ivankar

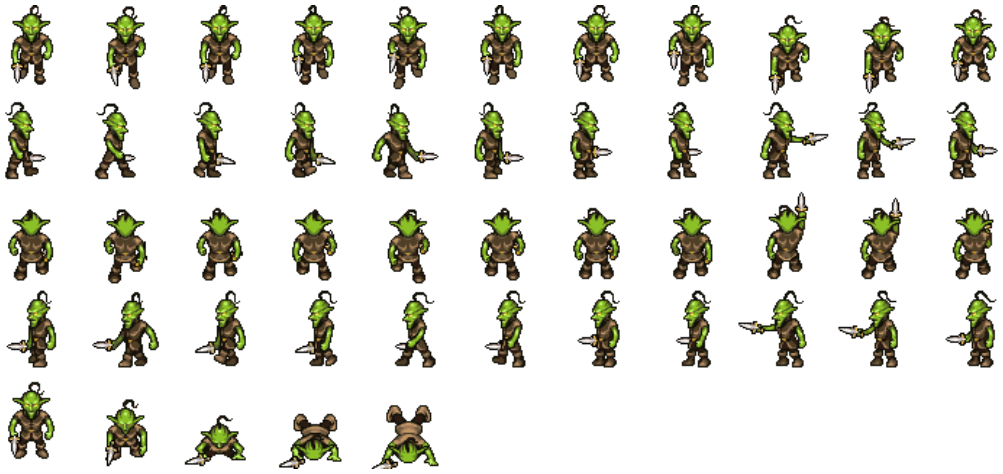


Figura 61- Sritesheet del enemigo Goblin

Figura 62- Spritesheet arañas (Enemigo Araña Negra)

Figura 63- Spritesheet monstruos (Enemigo esqueleto verde)

Figura 64- Imagen de fondo de batalla: Bosque

Figura 65- Spritesheet de Aldeano tipo 1

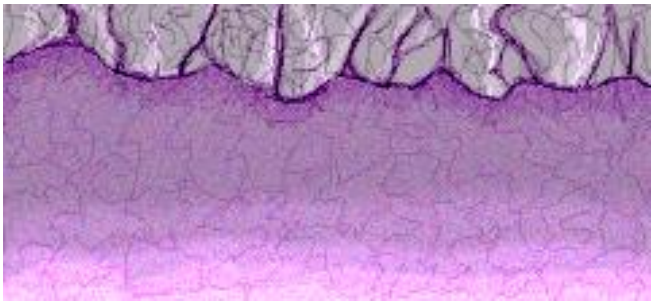


Figura 66- Fondo de batalla: Cueva

Figura 67- Spritesheet de Vendedor

Figura 68- Spritesheet de Aldeana tipo 2

Figura 69- Spritesheet Leana

Figura 70- Barriles

Figura 71- Spritesheet ciudad

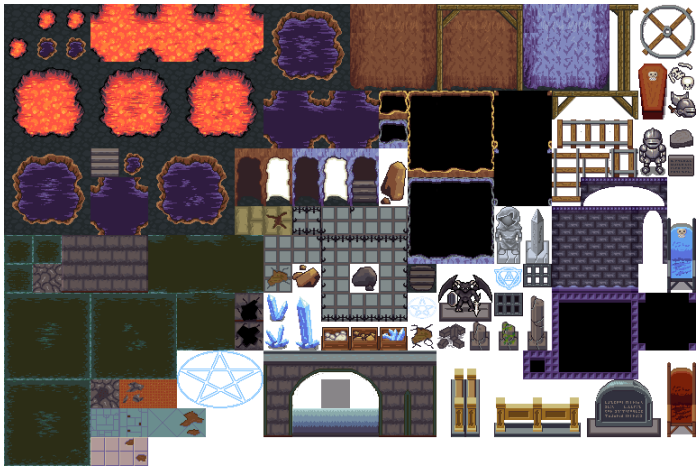


Figura 72- Spritesheet Cueva

Figura 73- Spritesheet para Exteriores Ciudad

Figura 74- Spritesheet para interior de las casas (1)

Figura 75- Spritesheet par interior de las casas (2)

Figura 76- Copas de árbol

Figura 77- Troncos de árbol

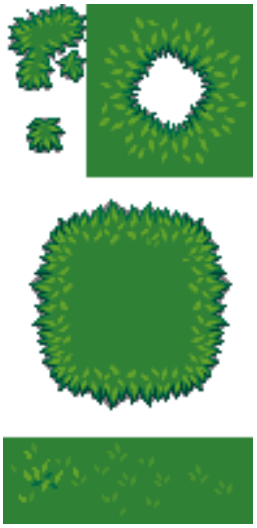


Figura 78- Terreno de hierba

Figura 79- Rocas

Figura 80- Retratos

6. Conclusiones

Este proyecto ha sido toda una experiencia donde me he dado cuenta de mis límites y carencias actuales.
He aprendido mucho a lo largo del desarrollo pues me he encontrado con muchos problemas ya que hubo situaciones en las que tuve que investigar mucho para poder resolverlos.
Algunos de estos problemas están relacionados con la implementación, ya que a la hora de programar no he tenido claro del todo como hacerlo y he tenido que buscar una gran cantidad de tutoriales que finalmente me han inspirado para seguir adelante. Otro gran problema que he tenido ha sido encontrar todos los recursos gráficos, pues estos iban a ser de terceros igualmente. Ha sido difícil no por cantidad de recursos 2D en la red, sino encontrar recursos 2D que encajaran con lo que yo quería y además que encajaran entre ellos para que hubiera cohesión artística dentro del juego.

Pese a que el resultado final no es el que tenía en mi mente, ya que en un principio la idea que llevaba en mi cabeza era distinta (sobre todo a nivel artístico), he podido completar los siguientes objetivos: hacer un combate por turnos con barra ATB, hacer un sistema de trabajos que modifica las estadísticas, movimiento del personaje, conversaciones con npcs, transición de niveles con efecto fade-in/fade-out y el uso de habilidades.

En conclusión, este proyecto es un paso más en mi formación, que marca una línea de salida para seguir aprendiendo y mejorando.

7. Anexo

7.1 Diagrama de clases

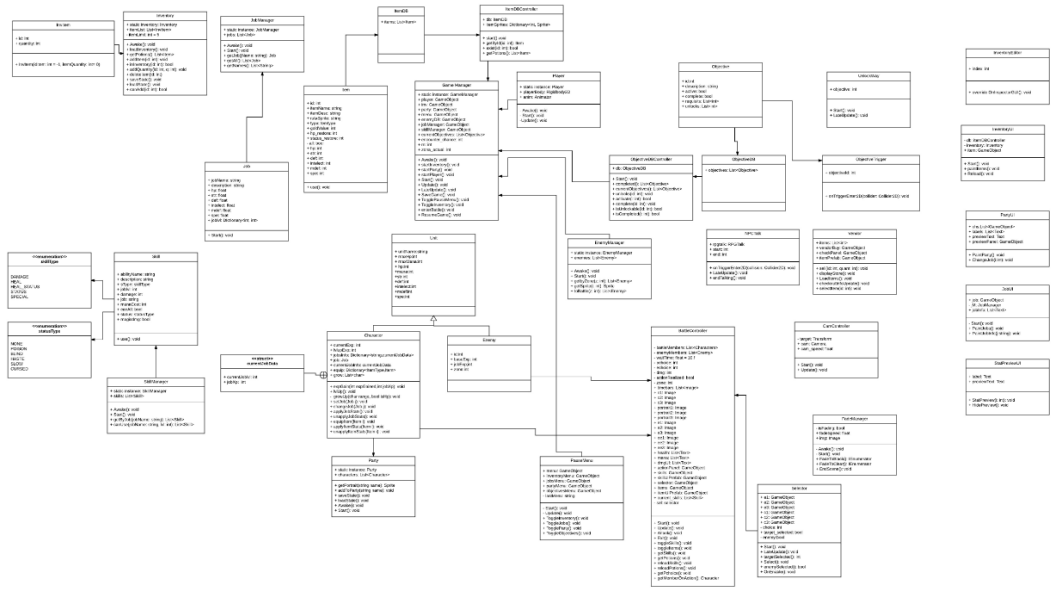


Figura 81- Diagrama de clases

7.1.1. Clase GameManager

Game Manager
+ static instance: GameManager + player: GameObject + inv: GameObject + party: GameObject + menu: GameObject + enemyDB: GameObject + jobManager: GameObject + skillManager: GameObject + currentObjectives: List<Objective> + encounter_chance: int + rn: int + zona_actual: int
+ Awake(): void + startInventory(): void + startParty(): void + startPlayer(): void + Start(): void + Update(): void + LateUpdate(): void + SaveGame(): void + TogglePauseMenu(): void + ToggleInventory(): void + enterBattle(): void + ResumeGame(): void

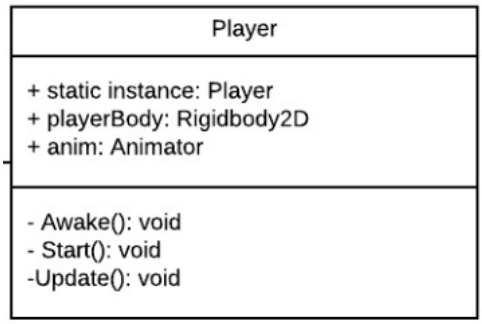
Esta es la clase que controla el flujo del juego, es decir es la que se encarga de almacenar y mantener la información general del juego (% de encuentros, zona actual, referencias a las demás entidades independientes del juego…) además de cambiar el estado del juego (mapa, pausa, batalla…).

Entrando más en detalle, los métodos pertenecientes a la clase son:

- Awake(): método que se llama al principio de la ejecución, donde se crea la instancia del propio GameManager como de otros singleton mediante la llamadas a los métodos pertinentes, así como crear una instancia de las variables GameObject enemyDB y skillManager.
- StartInventory(): método que instancia el GameObject “inv” creando así una instancia para la clase Inventory.
- StartParty(): método que instancia el GameObject party creando así una instancia para la clase Party.

- StartPlayer(): método que instancia el GameObject player creando así una instancia para la clase Player.
- Start(): método que inicializa algunas de las variables de la clase como encounter_chance o zona_actual.
- Update(): método que se encarga de controlar los cambios de estado, principalmente se encarga de comprobar si se entra o no en batalla.
- LateUpdate(): método que se encarga principalmente de recoger entradas de teclado.
- SaveGame(): método que llama a las funciones de salvado de algunas clases como Inventory.
- TogglePauseMenu(): método que pone el juego en pausa y muestra el menú de pausa.
- ToggleInventory()
- EnterBattle(): método que se encarga de cargar la escena perteneciente a la batalla.
- ResumeGame(): método que en caso de haber pausado el juego devuelve el juego al estado normal.

7.1.2. Clase Player

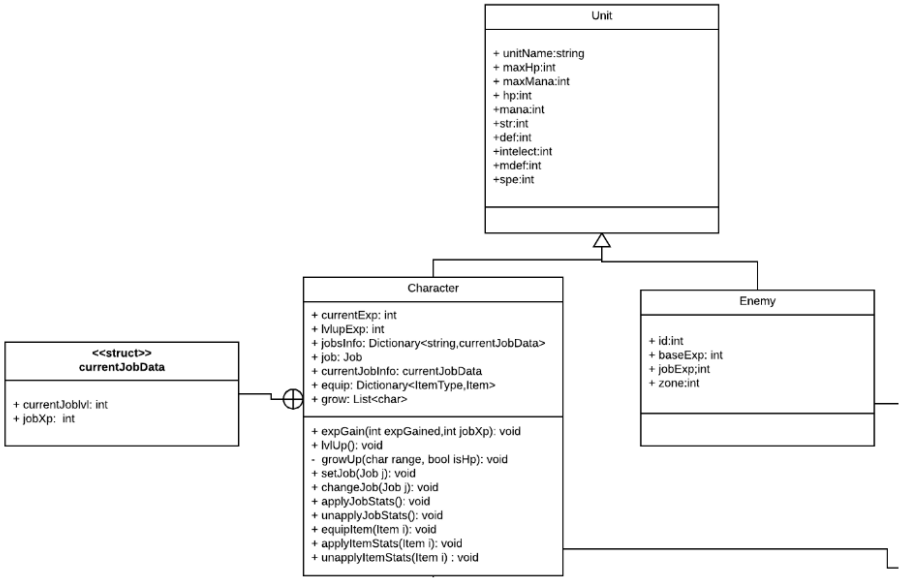


Esta clase, asociada a un sprite, se encarga de controlar el movimiento del jugador con las animaciones pertinentes.

Los métodos de esta clase son:

- Awake(): método donde se crea la instancia de la clase, ya que se un singleton.
- Start(): método en el que se realiza la inicialización de las variables playerBody y anim.
- Update(): método en el que se establecen los cambios de sprite del animator y además se modifica la posición del sprite al cual está asociada la clase.

7.1.3. Clases Unit, Character y Enemy



Estas 3 clases van prácticamente de la mano ya que la clase Unit es la clase base para las otras dos y se encarga de definir las estadísticas comunes tanto en los character como en los enemy, es decir, las estadísticas de vida, maná, fuerza, defensa, intelecto, defensa mágica y velocidad.

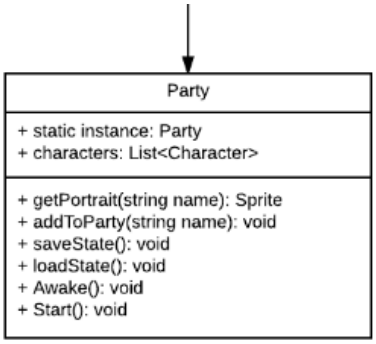
Por otro lado, tenemos la clase Enemy que hereda de la clase Unit, y tiene como atributos propios: id (para la base de datos de enemigos), la experiencia (para el personaje y para el trabajo) que da al derrotarlo y la zona a la que pertenece.

Finalmente, tenemos la clase Character, que define a los personajes jugables del juego, y además de la información de la clase Unit, almacena información propia como un registro de los trabajos que ha tenido el personaje (con nivel y experiencia en el momento), el trabajo actual equipado con su información de experiencia correspondiente, la lista de objetos que tiene equipados, las estadísticas modificados y una lista que define el crecimiento de estadísticas que va a tener el personaje.
Respecto a los métodos de la clase Character:

- ExpGain(int expGained, int jobXp): método que se encarga de sumar la experiencia obtenida a la experiencia actual y comprobar cuando se sube de nivel. Se aplica tanto a la experiencia propia del personaje como a la experiencia del trabajo que tiene asociado.
- lvlUp(): método que se encarga de la subida de nivel del personaje, actualizando así la experiencia necesaria para subir al siguiente nivel además de actualizar las estadísticas del personaje mediante la llamada el método growUP.
- growUp(char range, bool isHP): método que se encarga de aplicar cuanto crecen las estadísticas en función del rango especificado por parámetro.
- setJob(Job j): método que se encarga de asignar un trabajo al personaje.
- changeJob(Job j): método que se encarga de realizar el cambio de trabajo. Además guarda la información del trabajo anterior, recupera, en caso de existir, la información del nuevo trabajo equipado y actualiza las estadísticas del personaje acorde al nuevo trabajo.
- applyJobStats(): método que se encarga de aplicar los multiplicadores del trabajo a las estadísticas del personajes.
- unapplyJobStats(): método que se encarga de volver las estadísticas del personaje a su valor base.

- equipItem(Item i): método que se encarga de añadir a la lista de equipo el ítem pasado por parámetro.
- applyItemStats(Item i): método que se encarga de aplicar los aumentos de estadísticas que proporciona el objeto pasado por parámetro.
- unapplyItemStats(Item i): método que se encarga de volver las estadísticas a su estado anterior quitándole las mejoras del objeto pasado por parámetro.

7.1.4. Clase Party

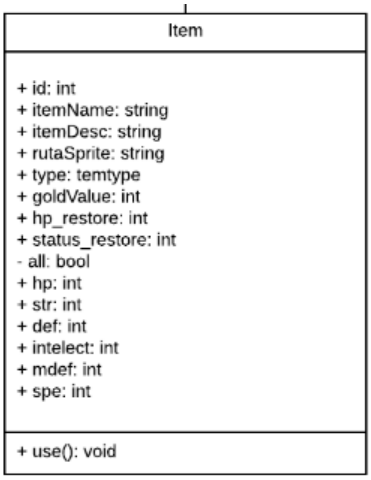


La clase Party es otra de las clases singleton y se encarga de almacenar los personajes jugables.

Los métodos de esta clase son:

- Awake(): método que crea la instancia de la clase.
- Start(): método donde se almacenan los personajes en la lista después de ser inicializada.
- GetPortrait(string name): método que devuelve el retrato del personaje que tenga por nombre el string pasado por parámetro.
- SaveState(): método que guarda en un fichero la información del grupo.
- LoadState(): método que recupera la información del grupo proveniente del fichero donde están guardados los datos.

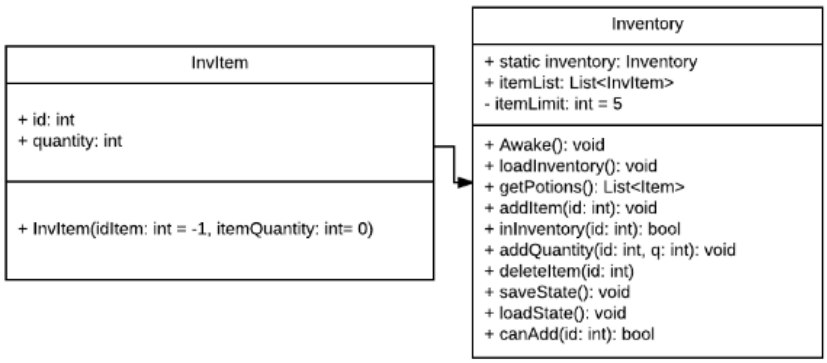
7.1.5. Clase Item



La clase Item define como va a ser un objeto en el juego, es decir, si va a ser una poción o un objeto equipable… Esta diferencia viene principalmente dada por la variable type.

El método use(), está pensado para las pociones y proporciona la información necesaria al BattleController para aplicar la curación pertinente.

7.1.6. Clase Inventory

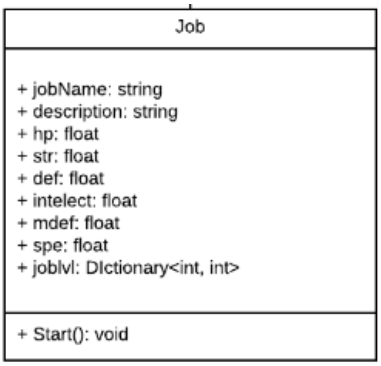


La clase inventario almacena los objetos que están en posesión del jugador, para ello se ha creado la clase InvItem que contiene un ID con el que se buscara el objeto en la base de datos y la cantidad que se tiene en posesión.

Los métodos que componen la clase son:

- Awake(): método que se encarga de crear la instancia de la clase.
- AddItem(int id): método que añade un objeto al inventario en función de su id (se comprueba que existe en la base de datos) en caso de que pueda añadirse (no se ha alcanzado el límite del inventario).
- AddQuantity(int id, int quantity): método que se encarga de buscar si el objeto cuyo ID es pasado por parámetro se encuentra dentro del inventario y en caso de estar dentro y no superar la cantidad de 99, se procede a aumentar la cantidad del objeto.
- InInventory(int id): método que busca el objeto en el inventario y devuelve verdadero o falso en función de si se encuentra o no.
- CanAdd(): método que devuelve verdadero o falso en caso de que puedan añadirse más objetos o no al inventario.
- GetPotions(): método que devuelve aquellos objetos en el inventario que son pociones.
- DeleteItem(int id): método que elimina del inventario el objeto especificado por parámetro en caso de estar contenido en el.
- SaveState(): método que guarda la información del inventario en un fichero json.
- LoadState(): método que recupera la información del fichero json para guardarla en la lista de objetos del inventario.

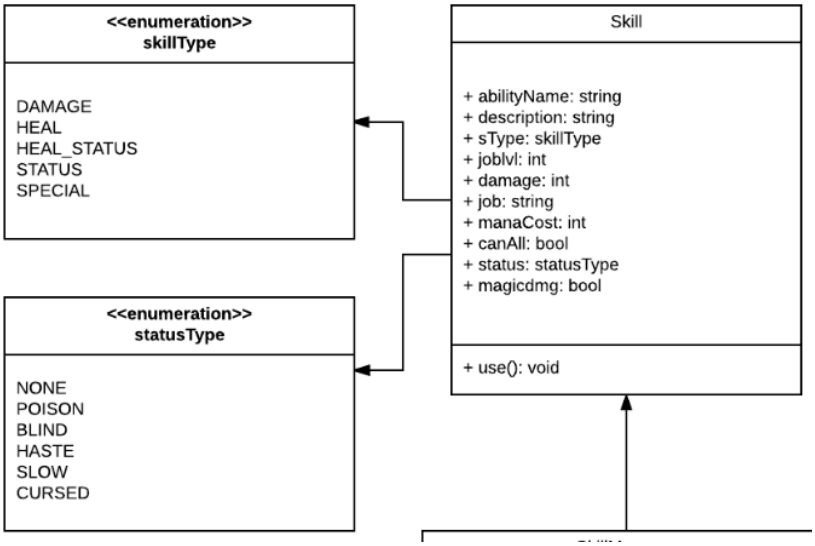
7.1.7. Clase Job



La clase Job es la que define las estadísticas de cada trabajo, es decir, los multiplicadores que afectan a las estadísticas de los personajes.

En su único método, Start(), se almacena la información en el diccionario joblvl, donde se establece la relación entre el nivel de trabajo y la experiencia necesaria para llegar al siguiente nivel de trabajo.

7.1.8. Clase Skill

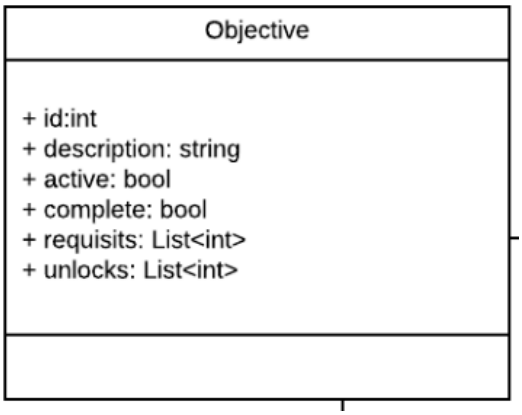


La clase Skill define las habilidades que podrán ser utilizadas por los personajes a través de los trabajos, ya que cada habilidad está directamente relacionada con un trabajo concreto.

La clase está diseñada para que se puedan crear habilidades con efectos distintos los cuales vienen definidos por el tipo de la habilidad (skillType), el tipo de estado en caso de que la habilidad añada un efecto de estado (statusType) y el booleano que indica si es daño mágico.

El método use() proporciona información al BattleController para aplicar el daño o curación pertinente.

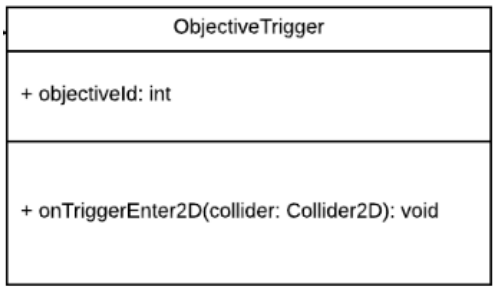
7.1.9. Clase Objective



La clase Objective define los objetivos que tiene el juego. Está pensado para que sea una cadena de objetivos/eventos donde un objetivo puede depender o no de un objetivo anterior y que un objetivo pueda desbloquear o no varios objetivos. Esto incluye también el que un objetivo puede depender de 1 o más objetivos para ser desbloqueado.

Un objetivo se desbloquea, en caso de tener un predecesor, una vez se han completado todos aquellos objetivos que están presentes en la lista de requisitos.

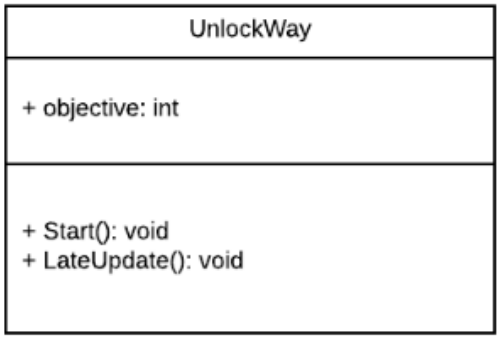
7.1.10. Clase ObjectiveTrigger



La clase ObjectiveTrigger está pensada para completar objetivos que requieran pasar por cierto lugar o llegar a cierto sitio.

El método onTriggerEnter2D se encarga de llamar internamente el controlador de objetivos y completar el objetivo en el caso de haber pasado por el lugar correspondiente.

7.1.11. Clase UnlockWay



La clase UnlockWay está pensada para desbloquear caminos en función de si un objetivo determinado se ha completado o no.

Los métodos de esta clase son:

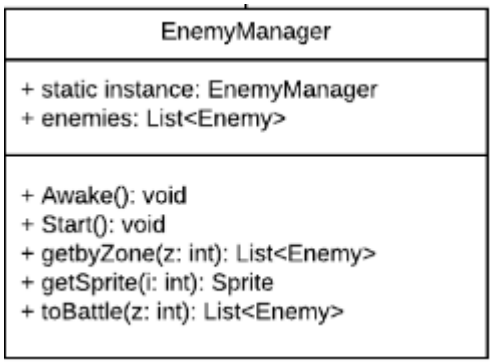
- Start(): método que comprueba durante la inicialización de la clase si el objetivo correspondiente ha sido completado o no, y en consecuencia eliminar el GameObject asociado o no hacer nada. Esto es debido a que hay objetivos que se completan en una escena diferente a donde está el camino bloqueado.

- LateUpdate(): método que al final de cada frame comprueba si el objetivo ha sido completado o no, y en caso de estar completado, elimina el GameObject al cual está asociado la clase.

7.1.12. Clases que actúan como bases de datos

Las siguientes clases actúan como bases de datos, es decir, como biblioteca de datos que está siempre disponible para extraerlos cuando es necesario.
Con la excepción de ItemDBController y ObjectiveController, el resto de bibliotecas no están hechas mediante información de fichero, sino que está realizada a mano en Unity mediante prefabs.

7.1.12.1 Clase EnemyManager



EnemyManager representa la biblioteca de enemigos del juego donde se almacenen en una lista todos los enemigos.

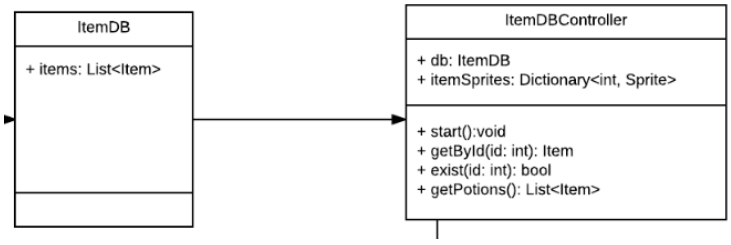
Respecto a los métodos Awake() y Start():

- Awake(): método que genera la instancia de la biblioteca.
- Start(): método que, al ser una instancia de un prefab de prefabs, recorre todos los GameObject que están asociados a la instancia y a partir de ahí recoge el script Enemy en cada uno de ellos y guardar así los enemigos en la lista.

Y como toda base de datos, la principal función de esta clase es devolver información, en este caso en forma de búsquedas o filtros:

- getbyZone(int z): devuelve todos los enemigos que se encuentran en la zona especificada con el int pasado por parámetro.
- getSprite(int i): devuelve el sprite del enemigo cuyo ID coincida con el int pasado por parámetro.
- toBattle(int z): devuelve una lista de 3 enemigos escogidos al azar de entre todos los que están en la zona especificada por parámetro.

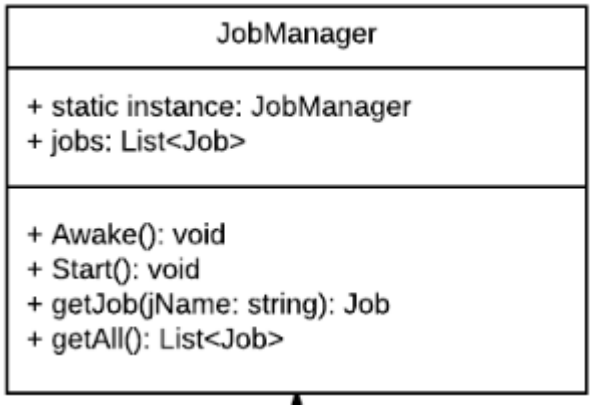
7.1.12.2. Clase ItemDBController



Esta biblioteca se genera mediante la lectura de un fichero json que contiene la información de todos los objetos del juego.

Su método Start() es el encargado de leer el fichero y almacenar toda la información, incluidos los sprites que se cargan en base a la ruta que hay en el fichero.

7.1.12.3. Clase JobManager



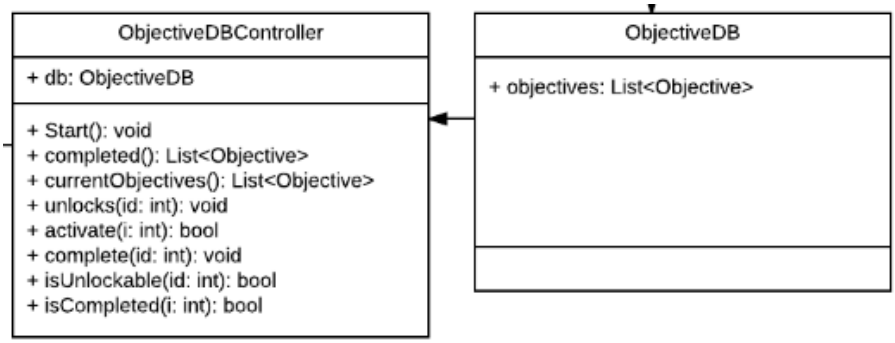
La clase JobManager representa la biblioteca de trabajos disponibles en el juego. La biblioteca es un prefab donde en el primer nivel esta el script de la clase y en los niveles inferiores están los GameObject que representan los trabajos.

Mediante los métodos Awake() y Start() se crea la estancia de la clase y se recorren los niveles inferiores o hijos del prefab para recoger el script Job asociado a cada uno de ellos.

Respecto al resto de métodos:

- getJob(string jName): método que devuelve el trabajo pertinente en función del nombre pasado por parámetro.
- getAll(): método que devuelve una lista con todos los trabajos disponibles.
- getNames(): método que devuelve una lista con los nombres de los trabajos disponibles.

7.1.12.4. Clase ObjectiveDBController

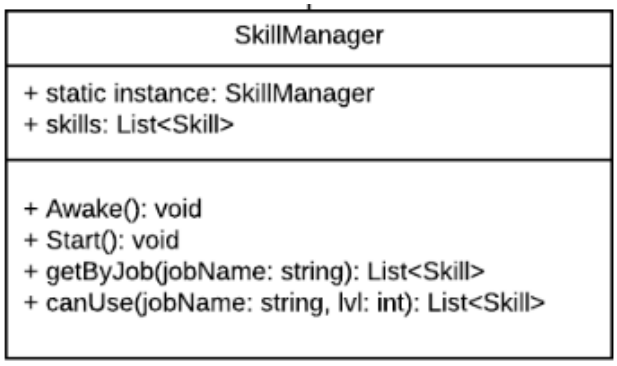


La clase ObjectiveController contiene todos los objetivos que estarán presentes durante el juego. La información de los objetivos se obtiene mediante la lectura de un fichero json.
Así pues la clase se encarga de llevar control sobre los objetivos que hay en curso y se encarga de completar y activar los objetivos pertinentes.

Los métodos de la clase ObjectiveController son:

- Start(): método en el que se inicializa la lista de objetivos mediante la lectura de datos del fichero json.
- Completed(): método que devuelve una lista de todos los objetivos que ya han sido completados.
- CurrentObjectives(): método que devuelve una lista con los objetivos que están activos.
- Unlocks(int id): método que en caso de ser posible, desbloquea todos los objetivos que dependen del objetivo pasado por parámetro.
- Activate(int id): método que activa el objetivo pasado por parámetro.
- Complete(int id): método que marca como completado el objetivo que se le pasa por parámetro.
- IsUnlockable(int id): método que comprueba si el objetivo pasado por parámetro es desbloqueable o no. Para ello comprueba si sus requisitos han sido completados.
- IsCompleted(int id): método que comprueba si el objetivo pasado por parámetro esta completado o no.

7.1.12.5. Clase SkillManager



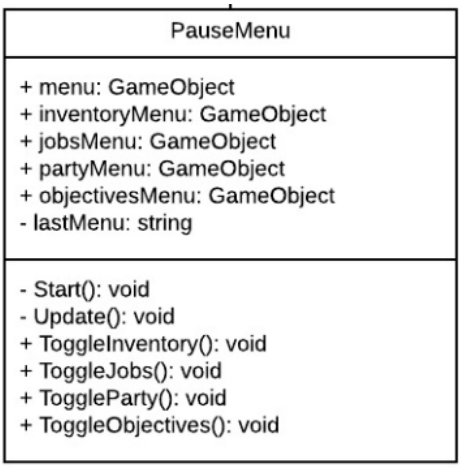
La clase SkillManager tiene el mismo funcionamiento que las clases JobManager y EnemyManager. Almacena en la lista de habilidades que tendrán los trabajos.

Con los métodos Awake() y Start() se crea la instancia de la clase y se recorren los hijos buscando los script de tipo Skill para guardarlos en la lista.

El resto de métodos son de tipo filtro:

- getByJob(string jobName): método que devuelve el trabajo que coincide con el nombre pasado por parámetro.
- canUse(string jobName, int lvl): método que devuelve todas las habilidades que se pueden utilizar con el nivel de trabajo especificado.

7.1.13. Clase PauseMenu

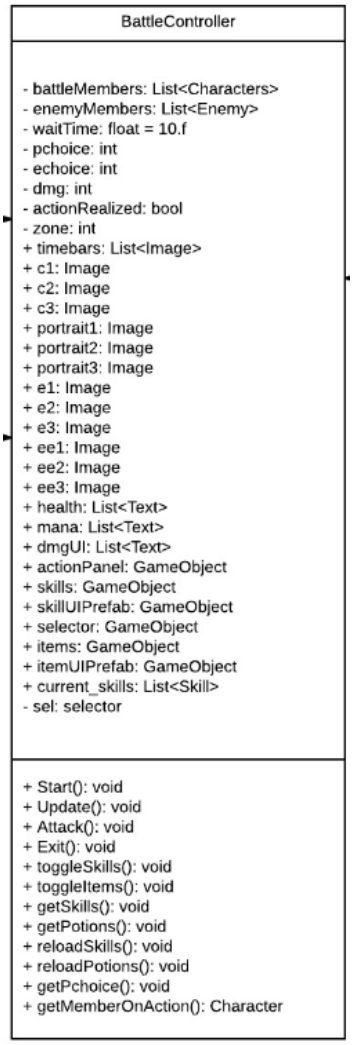


Esta clase se encarga de mostrar los distintos menús de información al usuario mediante los accesos a estos con las variables que hacen referencia a dichos menús. Así mismo también controla el flujo de navegación entre los distintos submenús.

Los métodos de esta clase son:

- Start(): método que se encarga de inicializar el menú principal (variable menú) como inactivo para que no se muestre en pantalla, así como preparar la navegación por los submenús inicializando la variable lastMenu.
- Update(): método que detecta si se quiere volver un paso atrás mediante input de teclado.
- ToggleInventory(): método que activa el menú de inventario y actualiza la navegación indicando que este menú es el último visitado.
- ToggleJobs(): método que activa el menú de trabajos y actualiza la navegación indicando que este menú es el último visitado.
- ToggleParty(): método que activa el menú de grupo y actualiza la navegación indicando que este menú es el último visitado.
- ToggleObjectives(): método que activa el menú de objetivos y actualiza la navegación indicando que este menú es el último visitado.

7.1.14. Clase BattleController



La clase BattleController se encarga de controlar todos los aspectos de la batalla, desde la interfaz hasta todos los cálculos de daño.

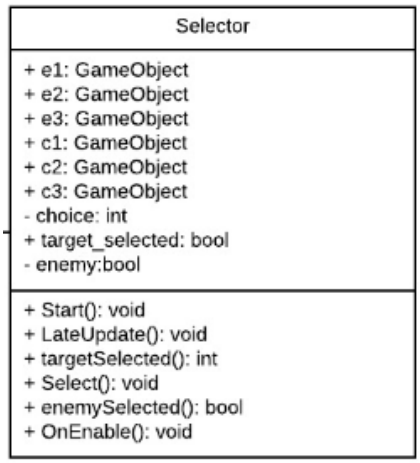
Los métodos de esta clase son:

- Start(): método que inicializa todas las variables, entre las que se encuentran: las unidades que participan en la batalla (tanto aliadas como enemigas) y los componentes de interfaz (sprites de enemigos y retratos de los personajes se enlazan a su correspondiente componente de interfaz).
- Update(): este método es el más importante pues es el que se encarga de controlar todo el flujo de la batalla. Se divide en distintos estados: espera, elección del jugador, elección del enemigo, selección de objetivo, cálculo de daño, victoria y derrota.
  En el estado de espera se van llevando las barras de tiempo y en el momento se llena una, detecta cual es y pasa a elección de jugador o de enemigo en función de la barra llena.
  En el estado de elección del enemigo, el enemigo elige al personaje con menos vida y le ataca.
  En el estado de elección de personaje se abre el menú con las acciones que se pueden realizar (atacar, habilidades, objetos, escapar).

En el estado de selección de objetivo se muestra un marco selector que puede moverse a través de las unidades para seleccionar la unidad que va a ser objetivo del ataque, habilidad u objeto.
En el estado de cálculo de daño se mira primero que unidad es la atacante y cuál es la atacada, una vez hecho esto se calcula el daño aplicando la fórmula pertinente en función del tipo de ataque (físico o mágico). Con el daño ya calculado, se aplica la diferencia de la vida de la unidad atacada con el daño infringido, actualizando así la estadística de vida. A continuación, se llama a la co-rutina "DmgUI" para mostrar el daño realizado/recibido por pantalla.
En el estado victoria se suman todas las ganancias de experiencia y se aplican a los personajes, posteriormente el juego vuelve al mapa desde el cual se ha entrado en batalla.
En el estado derrota se produce cuando las unidades del jugador tienen la vida a 0. Con esto viene el fin de la partida y devuelve al jugador al menú principal.

- Attack(): método que se encarga de recoger que personaje hace la acción y posteriormente pasa al estado de selección de objetivo.
- Exit(): método que se encarga de devolver el juego al mapa desde el cual se ha entrado en batalla.
- ToggleSkills(): método que muestra las habilidades del personaje que tiene el turno.
- ToggleItems(): método que muestra los objetos (pociones) del inventario.
- GetSkills(): método que recoge las habilidades del personaje. Se utiliza para cargar las habilidades en la interfaz.
- GetPotions(): método que recoge las pociones del inventario. Se utiliza para cargar las pociones en la interfaz.
- ReloadSkills(): método que se encarga de recargar el panel de habilidades, es decir, elimina el contenido del panel.
- ReloadPotions(): método que se encarga de recargar el panel de objetos, es decir, elimina el contenido del panel.
- GetPchoice(): método que devuelve que personaje está eligiendo.
- DmgUI(): método que se llama internamente a modo de co-rutina, donde muestra el daño infringido a una unidad encima de esta durante un breve periodo de tiempo y después desaparece el número de la pantalla.

7.1.15. Clase Selector

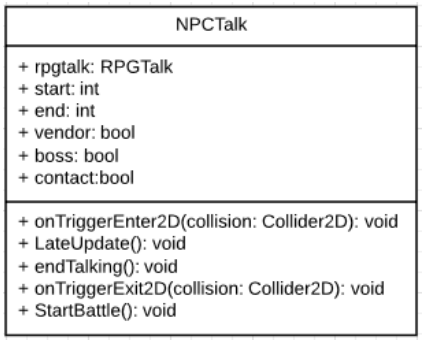


La clase Selector se utiliza en la batalla, cuando el jugador debe decidir el objetivo de su movimiento. De esta forma se muestra un marco que se puede mover por las posiciones de cada unidad.

Los métodos que componen la clase Selector son:

- Start(): método que inicializa las variables choice, target_selected y enemy. Además asigna la posición del selector a la posición donde se encuentra el primer objetivo (enemigo) disponible.
- LateUpdate(): método que controla la entrada de teclado del selector, haciendo que pueda moverse arriba, abajo, derecha e izquierda. Todo esto siempre y cuando sea un objetivo disponible.
- TargetSelected(): método que devuelve que objetivo se ha seleccionado en forma de índice.
- Select(): método que indica que se ha seleccionado un objetivo poniendo la variable target_selected a true.
- EnemySelected(): método que indica si el selector esta encima de un enemigo o no.
- OnEnable(): método que se activa cuando el selector pasa a estar activo y que se encarga de situar el selector donde este el primer objetivo (enemigo) disponible.

7.1.16. Clase NPCTalk



La clase NPCTalk esta enlazada a los NPC del juego y está pensada para que se produzcan las conversaciones. Todo gracias al asset RPGTalk. Las conversaciones se diferencian en 3 tipos: vendedor, jefe y cualquier npc, todo en función de las variables booleanas vendor y boss.

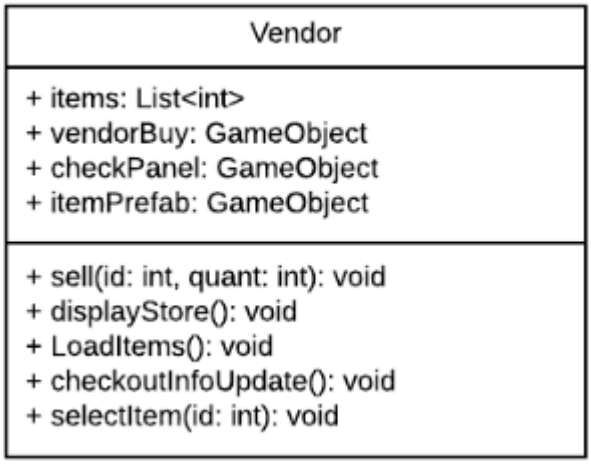
Los métodos de esta clase son:

- onTriggerEnter2D(Collider2D collision): método que al activarse gracias a la colisión del NPC con el jugador o viceversa, establece el principio y final de la conversación en la variable rpgtalk, dando lugar al inicio de la misma. El detalle especial del método está en que en función de que variable booleana este en verdadero (variables vendor o boss o ninguna), hará una llamada a un método distinto en cada caso al terminar la conversación. Además un cambio el estado del juego (pasa de estado MAP a estado TALKING) y pone la variable contact a verdadero.
- LateUpdate(): método que controla la entrada de teclado y es el encargado de, en caso de que se introduzca el input correcto, dar inicio a la conversación.
- EndTalking(): método que reestablece el estado del juego de vuelta a MAP al finalizar la

conversación.

- OnTriggerExit(): método que pone la variable contact a falso.
- StartBattle(): método que llama a la función enterBattle del GameManager.

7.1.17. Clase Vendor

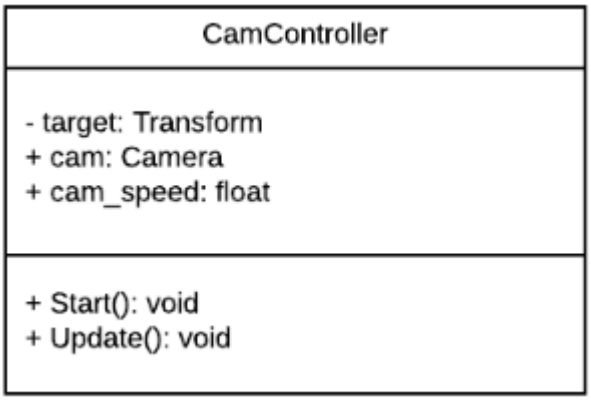


La clase Vendor, es la clase que almacenan y definen el comportamiento de los vendedores del juego.

La clase Vendor tiene los siguientes métodos:

- Sell(int id, int quant): método donde se añade el objeto con id igual a la del parámetro con la cantidad especificada mediante la llamada al método addItem del inventario, tras esto se actualiza el dinero que posee el jugador.
- DisplayStore(): método que se encarga de mostrar u ocultar la tienda.
- LoadItems(): método que se encarga de pintar los objetos que tiene en posesión el vendedor en pantalla.
- CheckoutInfoUpdate(): método que actualiza la información (cantidad y precio total de la compra) del objeto a comprar.
- SelectItem(int id): método que selecciona el objeto que se quiere comprar y actualiza la informacion en el panel de checkout (donde se muestra la cantidad de objetos a comprar y el precio total de la compra).
- ReloadItems(): método que limpia el panel donde se muestran los objetos para posteriormente poder mostrarlos correctamente al volver a mostrar la tienda.

7.1.18. Clase CamController

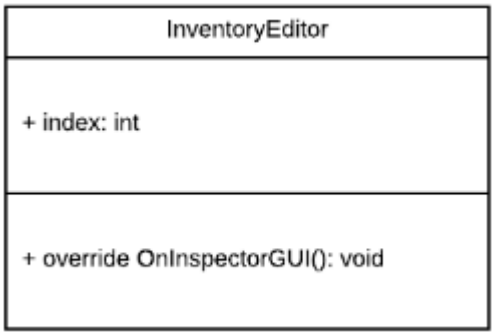


La clase CamController se encarga del movimiento de la cámara, que estará siguiendo al personaje controlado por el jugador en todo momento.

Los métodos de esta clase son:

- Start(): método que se encarga de inicializar las variables cam y target con los valores de la cámara y la instancia de la clase Player respectivamente.
- Update(): método que se encarga de establecer el tamaño de la cámara y de actualizar la posición de ésta mediante interpolación con la posición de la variable target.

7.1.19. Clase InventoryEditor



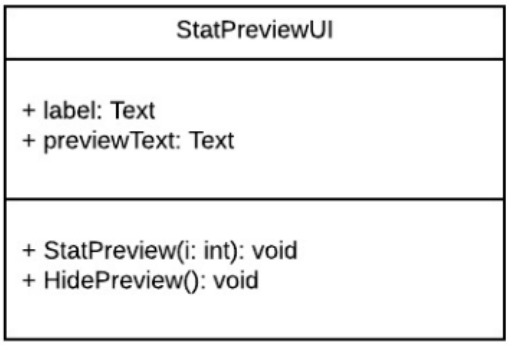
La clase InventoryEditor se encarga de modificar el inspector añadiendo etiquetas y controles para modificar el inventario del jugador. Es una clase destinada a realizar pruebas.

En su método OnInspectorGUI(), se recogen primeramente los objetos que hay en el inventario, se prepara un desplegable mediante un array de string y se crea un diccionario (con clave opción y de valor objeto) para guardar la información. A continuación se rellena el diccionario con la información de los objetos del inventario y se copian las claves en el array de string creado anteriormente.
Una vez hecho esto, se crea un campo para etiquetas donde se mostrarán los objetos y se rellena dicho campo mediante un bucle que recorre todos los objetos que hay en el inventario.
Seguidamente se añade al campo de etiquetas la lista desplegable aprovechando la variable index mediante la instrucción:

```
EditorGUILayout.Popup(index, options);
```

Finalmente se añaden dos botones para quitar y añadir objetos al inventario, los cuales llaman al método deleteItem y addItem respectivamente para poder realizar la acción.

7.1.20. Clase StatPreviewUI

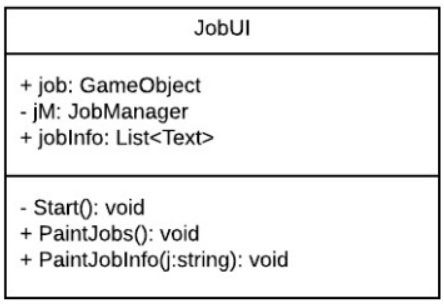


La clase StatPreviewUI se encarga de hacer una vista preliminar de las estadísticas a la hora de cambiar un trabajo de alguno de los personajes.

Los métodos de esta clase son:

- StatPreview(int i): método que se encarga primeramente, de activar el panel donde se muestra la vista previa. A continuación recoge las estadísticas base del personaje y los multiplicadores del trabajo del cual se hace la vista previa. Seguidamente, se empieza a formar la cadena de texto que irá dentro del panel de vista previa, donde mediante comparaciones entre la estadística modificada (actual) del personaje y la estadística de vista previa, se define si el texto sigue igual o es de color rojo (en caso de ser menos la estadística de vista previa) o verde (en caso de ser mayor).
- HidePreview(): método que se encarga de ocultar el panel de vista previa de estadísticas.

7.1.21. Clase JobUI

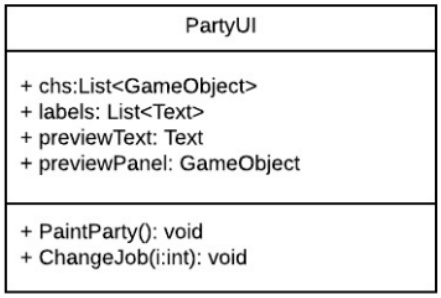


La clase JobUI es utilizada en el menú de trabajos, para poder mostrar toda la información en pantalla. Para ello la clase necesita de la variable job (que instanciará el prefab de interfaz de trabajo).

Los métodos de esta clase son:

- Start(): método que inicializa la variable de tipo JobManager y llama al método PaintJobs().
- PaintJobs(): método que se encarga de instanciar tantas veces como trabajos haya el prefab de interfaz de trabajos actualizando la información pertinente de cada instancia. Esta información es, el texto correspondiente con el nombre del trabajo y la información del componente botón añadiendo en su método onClick() la llamada al método PaintJobInfo.
- PaintJobInfo(string j): método que se encarga de pintar la información de un trabajo en concreto (nombre, descripción y modificadores). Esta información estará contenida dentro de la lista de componentes tipo texto.

7.1.22. Clase PartyUI

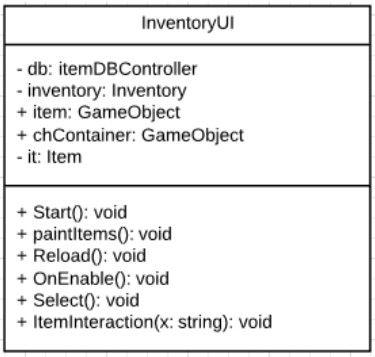


La clase PartyUI se encarga de mostrar en pantalla la información del menú de grupo, para ello recoge los paneles contenedores de los personajes en la lista chs y el panel de vista previa, además de algunos textos que facilitan el flujo de información.

Los métodos de esta clase son:

- PaintParty(): método que se encarga de poner la información de cada personaje dentro del panel que corresponde. Esto se hace recorriendo la lista de personajes del grupo y como los índices coinciden se convierte en tarea fácil el poder introducir la información.
- ChangeJob(int i): método que se invoca al cambiar el valor de la lista desplegable de alguno de los paneles de los personajes. Se pasa por parámetro en que panel se ha producido (y por tanto a que personaje se le cambia el trabajo) e internamente llama al método changeJob(string j) del personaje, siendo el string pasado por parámetro el texto con índice i de la lista etiquetas.

7.1.23. Clase InventoryUI



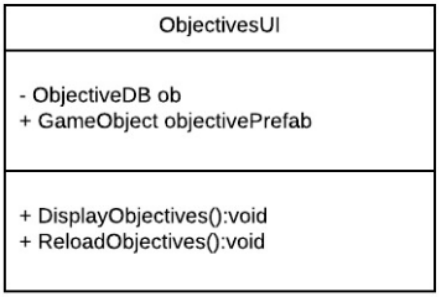
La clase InventoryUI se encarga de pintar por pantalla los objetos que hay en el inventario dentro del menú de inventario. Para ello instancia tantas veces como objetos haya el prefab de interfaz de objeto (alocado en la variable ítem).

Los métodos de esta clase son:

- Start(): método que inicializa las variables db e inventory y llama al método paintItems para pintar en primera instancia los objetos.
- PaintItems(): método que se encarga de instanciar tantas veces como objetos haya en el inventario el prefab ítem modificando la información de su texto por la del ítem correspondiente, previa llamada al método reload en caso de que ya haya objetos dentro del contendor. Esta información se obtiene de una consulta al inventario para que devuelva todos los objetos que hay dentro.
- Reload(): método que se encarga de vaciar el panel donde se almacenan todas las instancias del prefab para que al volver a pintar todo en pantalla no se duplique información. Esto se consigue recorriendo los hijos del panel (según el árbol jerárquico de la escena) y destruyéndolos uno a uno. Finalmente, llama al método paintitems para rellenar de nuevo el contenedor.

- OnEnable(): método que cada vez que se activa el GameObject llama al método reload para limpiar el contenedor de objetos del inventario.
- Select(): método que actualiza la variable it en función del objeto que se haya seleccionado.
- ItemInteraction(string x): método que en función del tipo de objeto seleccionado, aplica una curación al personaje de nombre x o bien equipa el objeto a dicho personaje.

7.1.24. Clase ObjectivesUI

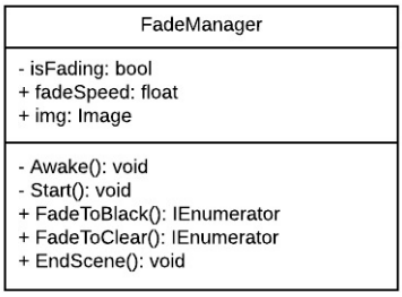


La clase ObjectivesUI se encarga de pintar por pantalla la información de los objetivos activos en ese momento dentro del menú de objetivos. Para ello instancia tantas veces como objetivos activos haya el prefab de interfaz de objetivo (alocado en la variable ob).

Los métodos de la clase ObjectivesUI son:

- DisplayObjectives(): método que se encarga de instanciar y colocar dentro del contenedor todas las instancias del predaf objetivo, aunque primeramente hace una llamada al método reloadObjectives en caso de que haya objetivos dentro del contenedor.
- ReloadObjectives(): método que se encarga de vaciar el contendor donde se almacenan todas las instancias del prefab objetivo. Esto se consigue recorriendo los hijos del panel (según el árbol jerárquico de la escena) y destruyéndolos uno a uno.

7.1.25. Clase FadeManager



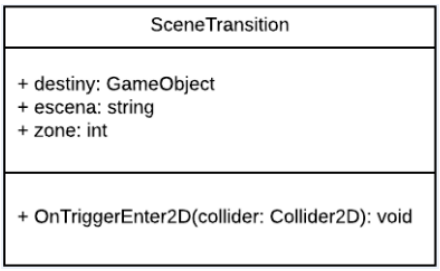
La clase FadeManager es la encargada de hacer el efecto de fade-in y fade-out al producirse un cambio de escena.

Los métodos de la clase FadeManager son:

- Awake(): método que se encarga de que el panel al que está asociado la clase no se destruya con los cambios de escena.
- Start(): método que se encarga de inicializar las variables de la clase.
- FadeToBlack(): método que se encarga de oscurecer la pantalla. Esto se produce aplicando una interpolación del color de una imagen de blanco a negro. Tras oscurecer la pantalla, se hace la llamada al método FadeToClear().

- FadeToClear(): método que, tras hacer una espera de 2 segundos, se encarga de aclarecer la pantalla. Esto se produce aplicando una interpolación del color de una imagen de negro a blanco.

7.1.26. SceneTransition



La clase SceneTransition es la clase encargada de realizar los cambios de escena y de mover la posición del jugador al punto de entrada de la siguiente escena.

Todo esto se produce dentro del método OnTriggerEnter(), donde se produce la llamada al método FadeToBlack de la clase FadeManager para empezar la transición, seguidamente se actualiza la posición del jugador (a la posición de la variable destiny), se cambia de escena (a la indicada en la variable) y se actualiza la zona en la clase GameManager.

8. Bibliografía

Estudio sobre el género RPG. Recuperado el 19 de Agosto de 2017, del sitio web de El Otro Lado: https://www.elotrolado.net/wiki/RPG#cite_note-0

Estudio sobre Unreal Engine. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://es.wikipedia.org/wiki/Unreal_Engine

Estudio sobre GameMaker. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://es.wikipedia.org/wiki/GameMaker:_Studio

Estudio sobre CryEngine. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://en.wikipedia.org/wiki/CryEngine

Estudio sobre Xenko Engine. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://en.wikipedia.org/wiki/Xenko

Estudio sobre RPGMaker. Recuperado el 19 de Agosto de 2017, de los sitios web Mundo-Maker, Profesionalreview y Wikipedia: http://www.mundo-maker.com/t12405-rpg-maker-mv-opiniones, https://www.profesionalreview.com/2016/02/25/analisis-del-rpg-maker-mv/, https://es.wikipedia.org/wiki/RPG_Maker

Videojuego Final Fantasy 6. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://es.wikipedia.org/wiki/Final_Fantasy_VI

Videojuego Final Fantasy 7. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://es.wikipedia.org/wiki/Final_Fantasy_VII

Videojuego Digimon World DS. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://en.wikipedia.org/wiki/Digimon_World_DS

Videojuego Suikoden 2. Recuperado el 19 de Agosto de 2017, del sitio web Wikipedia: https://es.wikipedia.org/wiki/Suikoden_II

Crecimiento de estadísticas. Recuperado el 19 de Agosto de 2017, del sitio web Suikosource: https://www.suikosource.com/games/gs2/guides/statgrowth.php

Trabajos Bravely Default. Recuperado el 19 de Agosto de 2017, del sitio web Neoseeker: http://www.neoseeker.com/bravely-default/faqs/817571-job.html

Experiencia de los trabajos. Recuperado el 19 de Agosto de 2017, del sitio web Guiasnintendo: http://www.guiasnintendo.com/0a_NINTENDO_3DS/bravely_second/bravely_second_sp/trabajos.html

Respositorio en Github. Recuperado el 19 de Agosto de 2017, del sitio web Github: https://github.com/broxigar91/TFG/

SeizeStudios. Recuperado el 19 de Agosto de 2017, del sitio web SeizeStudios: http://www.seizestudios.com/developer/rpgtalk/

Serialización en Unity. Recuperado el 19 de Agosto de 2017, del sitio web docs.unty3d.com: https://docs.unity3d.com/es/current/Manual/script-Serialization.html

Spritesheet Murcielago. Recuperado el 19 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/1-hour-lpc-enemy, créditos a Stephen "Redshrike" Challener.

Spritesheet Escarabajo. Recuperado el 19 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/lpc-beetle, créditos a Stephen "Redshrike" Challener.

Spritesheet Aldeana tipo 1. Recuperado el 19 de Agosto de 2017, del sitio web Untamed-wild.refuge: http://untamed.wild-refuge.net/rmxpresources.php?characters, créditos a Sithjester.

Spriteseet Axel. Recuperado el 19 de Agosto de 2017, del sitio web Untamed-wild.refuge: http://untamed.wild-refuge.net/rmxpresources.php?characters, créditos a Sithjester.

Spriteseet Ivankar. Recuperado el 19 de Agosto de 2017, del sitio web Untamed-wild.refuge: http://untamed.wild-refuge.net/rmxpresources.php?characters, créditos a Sithjester.

Spritesheet Goblin. Recuperado el 19 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/lpc-goblin, créditos a Stephen "Redshrike" Challener y WilliamThompsonj.

Spritesheet Arañas. Recuperado el 20 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/lpc-spider, créditos a Stephen "Redshrike" Challener y William Thompsonj.

Spritesheet Enemigos Radioactivos. Recuperado el 20 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/radioactive-rpg-monsters, créditos a Antifarea.

Fondo de batalla: Bosque y Cueva. Recuperado el 20 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/12-battle-backgrounds-240x110

Spritesheet Aldeano tipo 1. Recuperado el 20 de Agosto de 2017, del sitio web Untamed-wild.refuge: http://untamed.wild-refuge.net/rmxpresources.php?characters, créditos a Sithjester.

Spritesheet Vendedor. Recuperado el 20 de Agosto de 2017, del sitio web Untamed-wild.refuge: http://untamed.wild-refuge.net/rmxpresources.php?characters, créditos a Sithjester.

Spritesheet Aldeana tipo 2. Recuperado el 20 de Agosto de 2017, del sitio web Untamed-wild.refuge: http://untamed.wild-refuge.net/rmxpresources.php?characters, créditos a Sithjester.

Spritesheet Leana. Recuperado el 20 de Agosto de 2017, del sitio Opengameart: https://opengameart.org/content/lpc-sara, créditos a Stephen "Redshrike" Challener, William.Thompsonj y Mandi Paugh.

Spritesheet Copas de árbol, troncos de árbol, rocas, hierba y barriles. Recuperado el 21 de Agosto de 2017, del sitio Opengameart: https://opengameart.org/content/liberated-pixel-cup-lpc-base-assets-sprites-map-tiles, créditos a Lanea Zimmerman.

Spritesheet Ciudad. Recuperado el 21 de Agosto de 2017, del sitio Opengameart: https://opengameart.org/content/mage-city-arcanos, créditos a Hyptosis.

Spritesheet Exterior Ciudad. Recuperado el 21 de Agosto de 2017, del sitio Opengameart: https://opengameart.org/content/lpc-city-outside, créditos a Lanea Zimmerman, Hyptosis, Tuomo Untinen, Johannes Sjölund, Johann C y Daniel Eddeland.

Spritesheet Cueva. Recuperado el 21 de Agosto de 2017, del sitio Opengameart: https://opengameart.org/content/lpc-cavern-and-ruin-tiles, créditos a Lanea Zimmerman, William Thompson, Tuomo Untinen, Daniel Armstrong y Johannes Sjölund.

Spritesheet Interior de casas 1. Recuperado el 21 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/lpc-house-interior-and-decorations, créditos a Lanea Zimmerman, Tuomo Untinen, Daniel Eddeland, Manuel Riecke y Hyptosis.

Spritesheet Interior de casas 2. Recuperado el 22 de Agosto de 2017, del sitio web Opegameart: https://opengameart.org/content/lpc-city-inside, créditos a Lanea Zimmerman y Tuomo Untinen.

Spritesheet Retratos. Recuperado el 22 de Agosto de 2017, del sitio web Opengameart: https://opengameart.org/content/mostly-16x18-characters-and-48x48-portraits-repack, créditos a Charles Gabriel y Jorhlok.